Received 29 June 2022, accepted 12 August 2022, date of publication 22 August 2022, date of current version 30 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3200601

APPLIED RESEARCH

Security Countermeasures Selection Using the Meta Attack Language and Probabilistic Attack Graphs

WOJCIECH WIDEŁ, PREETAM MUKHERJEE, AND MATHIAS EKSTEDT
Division of Network and Systems Engineering, KTH Royal Institute of Technology, 10044 Stockholm, Sweden

Corresponding author: Mathias Ekstedt (mekstedt@kth.se)

This work was supported in part by the European Union's H2020 Research and Innovation Programme under Agreement 833481, and in part by the Swedish Civil Contingencies Agency through the research center Resilient Information and Control Systems (RICS).

ABSTRACT Connecting critical infrastructure assets to the network is absolutely essential for modern industries. In contrast to the apparent advantages, network connectivity exposes other infrastructure vulnerabilities that can be exploited by attackers. To protect the infrastructure, precise countermeasure identification is necessary. In this regard, the objective for the security officers is to identify the optimal set of countermeasures under a variety of budgetary restrictions. Our approach is based on the Meta Attack Language framework, which allows for convenient modelling of said infrastructures, as well as for automatic generation of attack graphs describing attacks against them. We formalize the problem of the selection of countermeasures in this context. The formalization makes it possible to deal with an arbitrary number of budgets, expressing available resources of both monetary and time-like nature, and to model numerous dependencies between countermeasures, including order dependencies, mutual exclusivity, and interdependent implementation costs. We propose a flexible and scalable algorithm for the problem. The whole methodology is validated in practice on realistic models.

INDEX TERMS Attack graphs, attack simulations, countermeasure selection, graphical security modeling, threat modeling.

I. INTRODUCTION

Well-being of today's organizations highly depends on their security posture. With more advanced attacking techniques devised by attackers with every passing day, organizations need to improve their security regularly. Security community at this juncture is striving for suitable methods to confidently assess security of a system and to find the actions which increase the level of security. During the last couple of decades, attack tree- and attack graph-based security analysis methods [2], [22], [25], [28], [34], [42] have been gaining popularity among security practitioners. By using these methods, it is possible to model multi-hop attacks in enterprise networks. With the help of various attack graph-based security metrics, viz. time-to-compromise [12], path length [14], attack difficulty [24], weakest adversary security metric [27], attack resistance [40], probabilistic security metric [39] it is possible to measure security of the target assets in a network. These metrics can be further used for suggesting actions by which security of valuable assets increases [20], [23], [25], [29], [35], [37].

Before an attack graph relating to an enterprise network (or an IT infrastructure in general) can be analyzed, it needs to be generated. Accuracy of the generated attack graphs depends on the collection of accurate information about the underlying IT infrastructure and its vulnerabilities. Different attack graph generation methodologies collect such information automatically from standard tools and databases [15], [26]. These methods are not perfect, and the accuracy and reusability of generated graphs is problematic [13], [36]. To tackle these issues, the Meta Attack Language (MAL) has been introduced in [16]. MAL is a model-driven security

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

engineering approach to support the development of domain specific languages (DSLs) for attack graph generation and attack simulations. With a DSL created by security experts in a particular domain, e.g., in SCADA systems for power industry, security non-experts can easily create models of their systems. From such a model and the security knowledge encoded in the DSL, attack graphs describing attacks on the system are generated automatically. With such attack graphs it is possible to analyze system architectures by for instance identifying weak spots in the design. However, normally an architecture features a multitude of weaknesses, more or less severe. From a design point of view the DSL enables easy, but trial-and-error-based, analysis. With large IT infrastructures, however, this approach naturally becomes labor-heavy.

A number of MAL-based DSLs have been already developed[1] for vehicles [18], [19], for energy sector [10], general-purpose DSL for IT infrastructures [17], etc.

In this work, we consider the problem of selection of countermeasures for improving security of an IT infrastructure. We use MAL as the tool for infrastructure modeling, and rely on MAL-generated attack graphs for assessing the infrastructure's security. Within this setting, we make the following contributions.

1) We formalize the problem of countermeasures selection, in a way that allows handling multiple budget constraints of different nature, as well as specifying various dependencies between the countermeasures.
2) We propose a highly parametrizable iterative algorithm solving this problem and discuss the algorithm's variants.
3) We perform experiments using a prototype implementation, illustrating the usability and scalability of our methodology.

In Section II, we discuss the existing works related to attack modeling and countermeasure selection. In Section III, we give a brief overview of the MAL framework. The problem of countermeasures selection is formalized in Section IV. We present our algorithm in Section V, and discuss its variants in Section VI. Empirical validation of our framework is provided in Section VII. Section VIII concludes the paper.

II. RELATED WORK

Suggesting ways of improving security of a system is one of the basic objectives of risk analysis. The method applied for its achievement will depend, among other factors, on the available model of the system and the employed security metrics. Because of this variability, a vast body of work tackling this issue exists.

In most of the existing approaches, it is assumed that available data describes the system and related risks completely. In such a case, it is possible to model and solve the problem using standard optimization techniques, such as mixed integer linear programming [5], [8], [20], [32], [45], stochastic programming [8], [45] or others. In [23] and [8], such data is obtained by extracting *all* attack paths from a model, which leads to exponential blow-up already in the case of moderately sized models. In the case of the MAL framework, real-life–sized models can easily contain hundreds of thousands of objects, and the attack graphs describing the complete possible behaviour space of both attacker and defender will have millions of nodes. Using all the information contained in the graph, not to mention extracting all attack paths, is thus not feasible in practice. We manage these difficulties by a repeated extraction of a reasonable number of optimal attack paths, which in turn is sufficient for producing beneficial countermeasures suggestions.

Selection of countermeasures is central to a number of modelling frameworks, such as attack-countermeasure trees [30], [31], protection trees [6] or attack-defense trees [21], to name just a few. Languages for specifying attack-defense trees are also developed [11]. Interested reader is referred to the survey [22] for an exhaustive overview of DAG-based security modelling, where also the aforementioned frameworks are presented. Recent developments in the field of attack-defense trees are described in detail in [42]. The main focus of the survey presented in [25] are precisely methods for optimal selection of countermeasures, including the ones based on attack trees and attack graphs. In most of the tree-based frameworks for modelling security (in contrast to the DAG-based ones) the causal or temporal dependencies between particular steps are not modelled explicitly, which makes the models in some sense less expressive than attack graphs. Furthermore, this fundamental difference makes it difficult, if not impossible, to adapt methods developed for such frameworks to the case of attack graphs.

Attack graphs are also being studied extensively and employed for the purpose of optimal selection of countermeasures, e.g., in [20], [29], [35], [37]. While the objects called attack graphs in [37] are essentially the same as the attack graphs of [1], and a reduced version of the latter, called *vulnerability dependency graph*s in [1] is used in [35], in each of the works [20], [29], [37] attack graphs are defined differently. Furthermore, each of them having different optimization goals, the optimization methods of [20], [29], [35], [37] require different input data. Together with the commonly shared assumption of the input model describing the security aspects of the system completely, this makes these methods not applicable in the case of MAL-based attack graphs.

A. DETAILED STUDY OF COUNTERMEASURE SELECTION METHODOLOGIES

The heuristic approach of countermeasure selection described in [37] uses the attack graph generation process of MuLVAL. The concept of pre-condition and post-condition used for attack steps are reused for mitigation actions, where post-conditions are the facts getting canceled by the implemented countermeasures. These countermeasures are practical techniques for implementing more generic mitigation actions. While calculating the risk, the likelihood of occurrence of an attack is computed from the existence of

[1] https://github.com/mal-lang

countermeasures and the CVSS access complexity metric. Finally, a heuristic-based approach is implemented to find out the set of countermeasures that minimizes the overall risk without exceeding the given budget.

Cost-Impact Countermeasure Selection is the attack countermeasure selection approach based on cost impact analysis [35]. This approach considers the costs of nodes unavailability, costs of different countermeasures, and recovery costs. Impact assessment graph, created by combining the attack graph and dependency graph, is used to compute the impact of an attack or a countermeasure implementation. The solution includes the dynamic nature of the attacker-defender game. The selection of countermeasure minimizes the service loss, countermeasure cost, and recovery cost.

In [29], authors put forward a Bayesian attack graph-based risk management approach. The local conditional probability distribution is defined to describe the chance of compromise of the asset vulnerabilities. Updated knowledge about attack incidents can be included in the model as the posterior probability. The issue of selection of the security mitigation plan is converted to the single and multi-objective optimization problems. The cost of security control implementation and expected loss/gain value related to the compromise of asset vulnerability are the factors involved while working on the optimization. A genetic algorithm-based solution is proposed to derive an efficient plan for implementation.

Countermeasure selection problem is described as a multi-objective min-max optimization problem in [20]. Probabilistic attack graph model is defined to develop the analysis. Both the attacker's and defender's perspectives are taken into account while formulating the optimization problem. Attacker wants to maximise the risk by selecting the most efficient attack path, whereas defender wants to minimize the risk. The min-max optimization problem for defender is converted to a single minimization problem by using the duality in linear programming.

In [45], authors described selecting a security mitigation techniques portfolio as a maximum vulnerability coverage problem with various practical constraints. Sets of attack paths to the target assets are assumed to be provided as input. Deterministic and stochastic models for mitigation coverage are proposed corresponding to the certainty (or uncertainty) of mitigation effectiveness. Group cardinality models describe the real-life constraints on mitigation implementation. Finally, different approximation-based algorithms are proposed to solve the problem of budgeted maximum multiple coverage and other derivatives.

Probabilistic attack-response trees, including attack steps (with probability), alerts (indicating attack incidents), and countermeasures, are defined in [23] to explain the countermeasure selection problem to stop multi-path attacks. The probability of an atomic attack is computed depending on the generated alerts. Suspicious attack paths are identified from the expected damage potential computed from the probability of atomic attacks, the CVSS score of the corresponding vulnerabilities, and the importance of the devices with vulnerabilities. ROI-based countermeasure performance score is used to find out the utility of the selected countermeasure. The proposed algorithm maximizes the response utility score calculated from the countermeasure-specific security benefit, deployment cost, and negative impact on service quality.

Attack–defense tree based formalism is used for selecting countermeasures in [8]. A computationally demanding procedure is employed for extracting rational behavior of the attacker and the defender from models, resulting in all possible attacks encoded in the model, as well as in minimal ways of blocking those attacks. The authors formulate a general integer linear programming program, as well as its stochastic variant, that employs the knowledge of the attacks for deriving set of countermeasures that is optimal in one of many possible senses, e.g., maximizes the number of countered attacks or prevents the attacks that are the most appealing for the attacker.

The cost optimization algorithms section described in the survey of attack graph analysis [44] is also worth mentioning here.

III. MAL FRAMEWORK

In this section, we provide a brief introduction to the MAL framework, sufficient for understanding of the remainder of the paper. We note that MAL constitutes background for the present work, accepted as is. For a more verbose intuitive description of MAL the reader is referred to [16] and to a web repository[2] for implementation support.

A. BRIEF INTRODUCTION TO MAL

The goal of MAL is twofold: to facilitate creation and reusage of IT infrastructure models, and to automate security analysis of such models. MAL allows for development of security-oriented domain specific languages (DSLs). The DSLs created with MAL revolve around two basic concepts: that of *assets* and *associations*, which are abstractions of types of objects and relationships between objects that arise in reality.[3] A MAL-created DSL might thus contain assets such as `Computer` and `Network`, as well as an association `Communication`, which will represent the ability of computers to communicate over networks. The domain-specific security knowledge is encoded in a DSL by assigning to assets *attack steps* and *defense steps*, along with a description of relations between them. The attack steps represent possible behaviour of an attacker relating to assets, and the defense steps – ways of hindering the attacker's actions.

Models consist of instantiations of assets and associations, called *objects* and *links*, respectively. A visualization of a toy model of an IT infrastructure created using a MAL-based DSL is given in Figure 1. The model is explained below.

[2]https://mal-lang.org/

[3]Similarly to class diagrams, e.g., as in the Unified Modelling Language (UML).

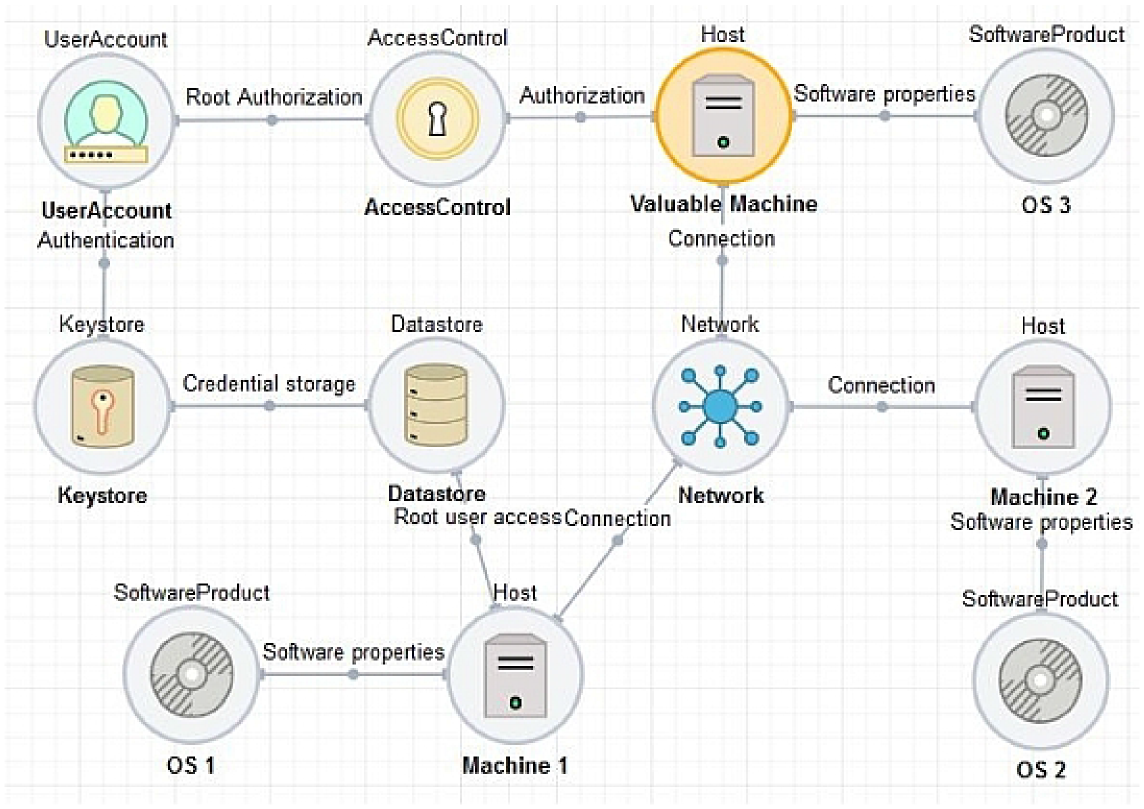

FIGURE 1. **A toy model of an IT infrastructure.**

Example 1: The circular icons in Figure 1 represent objects, and the line segments connecting them - links. Each object is labeled with two labels: the name of the object, placed below the icon, and the name of the asset that the object is an instance of, placed above the icon. Similarly, links are labeled with names of associations that they are instances of.

Thus, in the modelled infrastructure there are three `Hosts`: *Machine 1*, *Machine 2*, and *Valuable Machine*. There is a `Network` object, called *Network*, to which all three hosts are *connected* through a `Connection` link. Each of the hosts runs an operating system - the `SoftwareProduct` objects called *OS 1*, *OS 2* and *OS 3*. Finally, the computer *Machine 1* contains a `Datastore`, a part of which is a `Keystore` on which credentials allowing for root authorization by *UserAccount* on *Valuable Machine* are stored.

Security analysis of models within the MAL framework relies on the concept of *Time To Compromise* (TTC) and the effectiveness of defense steps. In a DSL, each of the attack steps is assigned a probability distribution[4] of time needed for its execution, and each of the defense steps – a Bernoulli distribution describing the probability of the modelled defense to be functional. The first step in the analysis is to unfold the model, by following the attack logic described by relations between steps in the DSL, obtaining as a result an *attack graph*.

[4]The distributions specified in a DSL can be seen as *defaults*; to fit the modelled infrastructure more appropriately, they can be modified on the model level.

B. MAL-DERIVED ATTACK GRAPHS

An attack graph generated from a model is a directed acyclic graph. Its nodes correspond to attack and defense steps that can be performed in the modelled system. The edges model the attack logic, as encoded in the DSL. An arc from an attack step v to an attack step w means that execution of v is a prerequisite for execution of w. Furthermore, attack steps are typed: there are AND attack steps and OR attack steps. An AND attack step can be performed only if all of its prerequisites have been executed; an OR attack step can be executed as soon as one of its prerequisites has been performed. An arc from a defense step u to an attack step w means that if the defense u is successfully implemented, then w cannot be executed, even if its prerequisites have been executed. Finally, each of the nodes in an attack graph is assigned a probability distribution originating from the model. With $\mathcal{F}$ denoting the set of all probability distributions, attack graphs considered in this work are defined as follows.

Definition 1 (Attack Graph): An attack graph is a tuple $G = (V, E, type, ttc)$, where

- (V, E) is a directed acyclic graph,
- $type\colon V \to \{\&, |, \#\}$ is a function assigning types to nodes in V, in such a way that none of the nodes of type $\#$ has a parent,

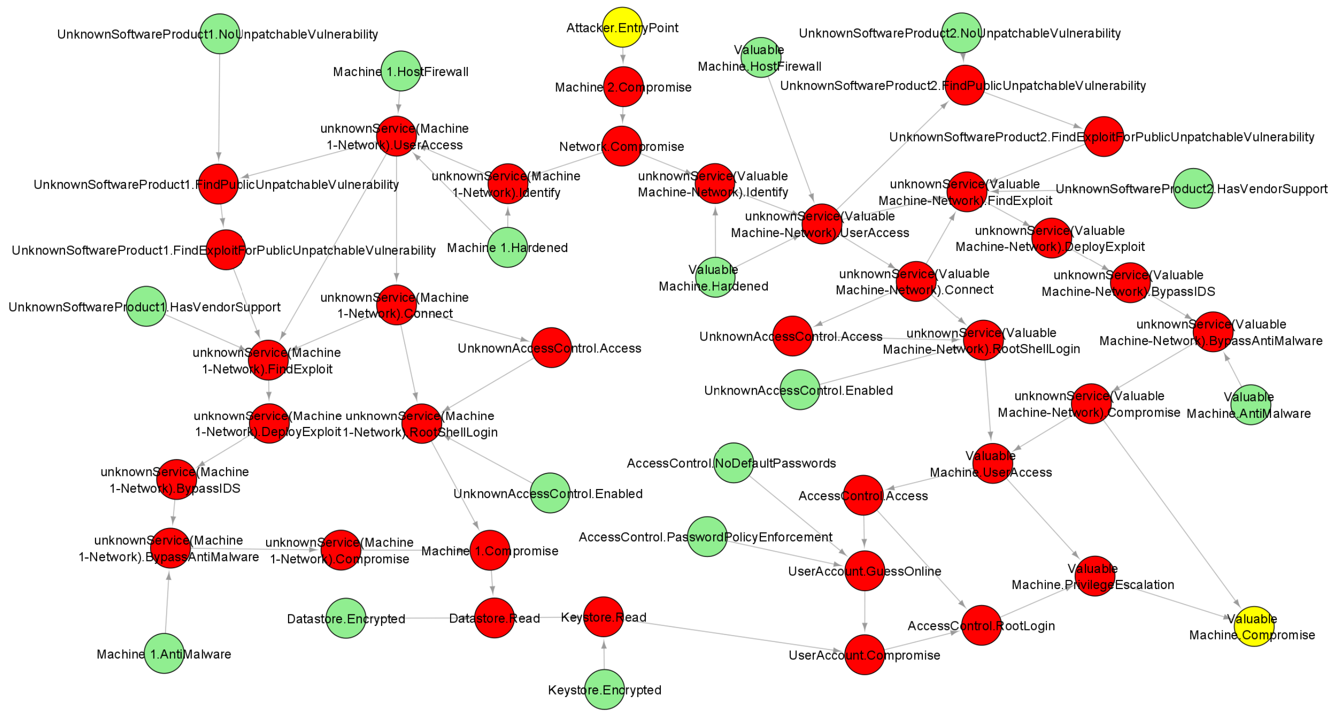
UnknownSoftwareProduct1.NoUnpatchableVulnerability
Machine 1.HostFirewall
Attacker.EntryPoint
Machine 2.Compromise
Network.Compromise
Valuable Machine.HostFirewall
UnknownSoftwareProduct2.NoUnpatchableVulnerability
UnknownSoftwareProduct2.FindPublicUnpatchableVulnerability
UnknownSoftwareProduct2.FindExploitForPublicUnpatchableVulnerability
unknownService(Machine 1-Network).UserAccess
UnknownSoftwareProduct1.FindPublicUnpatchableVulnerability
unknownService(Machine 1-Network).Identify
unknownService(Valuable Machine-Network).Identify
unknownService(Valuable Machine-Network).UserAccess
unknownService(Valuable Machine-Network).FindExploit
UnknownSoftwareProduct2.HasVendorSupport
UnknownSoftwareProduct1.FindExploitForPublicUnpatchableVulnerability
Machine 1.Hardened
unknownService(Valuable Machine-Network).DeployExploit
Valuable Machine.Hardened
unknownService(Valuable Machine-Network).Connect
unknownService(Valuable Machine-Network).BypassIDS
UnknownSoftwareProduct1.HasVendorSupport
unknownService(Machine 1-Network).Connect
unknownService(Machine 1-Network).FindExploit
UnknownAccessControl.Access
UnknownAccessControl.Access
unknownService(Valuable Machine-Network).RootShellLogin
unknownService(Valuable Machine-Network).BypassAntiMalware
UnknownAccessControl.Enabled
unknownService(Valuable Machine-Network).Compromise
Valuable Machine.AntiMalware
unknownService(Machine 1-Network).DeployExploit
unknownService(Machine 1-Network).RootShellLogin
AccessControl.NoDefaultPasswords
Valuable Machine.UserAccess
unknownService(Machine 1-Network).BypassIDS
UnknownAccessControl.Enabled
AccessControl.Access
unknownService(Machine 1-Network).BypassAntiMalware
unknownService(Machine 1-Network).Compromise
Machine 1.Compromise
AccessControl.PasswordPolicyEnforcement
UserAccount.GuessOnline
Valuable Machine.PrivilegeEscalation
Datastore.Encrypted
Datastore.Read
Keystore.Read
AccessControl.RootLogin
Valuable Machine.Compromise
UserAccount.Compromise
Machine 1.AntiMalware
Keystore.Encrypted

FIGURE 2. An attack graph depicting attacks on ***Valuable Machine*** in the model from Figure 1.

- *ttc*: $V \to \mathcal{F}$ is a function assigning probability distributions to nodes in V, in such a way that each of the nodes of type # is assigned a Bernoulli distribution.

The AND attack steps, OR attack steps and defense steps are assigned types &, | and #, respectively. The defenses in the current form of the MAL framework are not allowed to have parents, i.e., the possibility of the attacker countering a defense step or a defense step being a prerequisite for another defense step cannot be modelled.[5] Furthermore, for technical reasons, we require the attack steps having no parents to be assigned type &. The *ttc* abbreviation stands for *Time To Compromise*. Notation $ttc(v) = f$ means that the time needed for execution of the attack step v follows the distribution f.

For an attack graph G, we let $D_G = \{v \in V : type(v) = \#\}$ to be the set of all defense steps in G. Similarly, we set $A_G := V \setminus D_G$ to be the set of all attack steps in G. If an attack step has a defense step among its parent nodes, then it is said to be *counterable*.

An example of an attack graph is depicted in Figure 2. Defense nodes are represented with green circles, and attack nodes – with red circles. The nodes are labeled, with the labels being of the form *ObjectName.StepName*, with the names of objects coming from a model, and the names of steps – from the DSL used for creating the model. An example fragment from a DSL, in MAL syntax, aligned with generating this model is presented below.

```
asset Host {
| Compromise
  -> \ldots
& PrivilegeEscalation
  -> Compromise
| UserAccess
  -> PrivilegeEscalation,
     accessControl.Access,
     \ldots
}

asset AccessControl {
| Access
  -> rootAccess.attemptGuessDefualtPw,
     rootAccess.attemptGuessPolicyPw,
     \ldots
| RootLogin
  -> host.PrivilegeEscalation,
     \ldots
# NoDefualtPasswords
  -> rootUser.attemptGuessDefualtPw,
     \ldots
# PasswordPolicyEnforcement
  -> rootUser.attemptGuessPolicyPw,
     \ldots
}

asset UserAccount {
| Compromise
```

[5]Modelling of the attacker disabling or bypassing defenses is nevertheless possible, it requires simply introducing appropriate attack steps.

```
   -> rootAccess.RootLogin,
      \ldots
& attemptGuessDefualtPw
                 [Bernoulli 0.6] @hidden
   -> GuessOnline
& attemptGuessPolicyPw
                 [Gamma(25, 1)] @hidden
   -> GuessOnline
| GuessOnline
   -> Compromise
}


associations {
UserAccount [rootUser] *
<-- Root Authorization -->
 0..1 [rootAccess] AccessControl

Host [host] *
<-- Authorization -->
* [accessControl] AccessControl
\ldots
}
```

We note that the precise meaning of the steps is up to the creators of the DSL to define.

The relation between the graph in Figure 2 and the corresponding model in Figure 1 is more generally explained in Example 2.

Example 2: In the model from Figure 1, it has been assumed that the attacker has compromised *Machine 2*, and that the goal is to *Compromise Valuable Machine*. The entry point of the attacker and the target attack step are marked yellow in Figure 2. The fact that Machine 2 has been initially compromised is reflected by an arc connecting the steps *Attacker.EntryPoint* and *Machine 2.Compromise*, and by setting the TTC of both steps to 0. Note that each of the paths starting in the entry point ends in the node *Valuable Machine.Compromise*.

Two types of attacks can be identified in the graph. The paths traversing the left-hand side part of the graph correspond to the attacker exploiting vulnerabilities in a connection between *Network* and *Machine 1* to compromise the latter, getting access to the credentials (by executing the attack step *Keystore.Read*), and using the credentials for compromising the target machine. The attack paths in the right-hand side part of the graph represent attacks in which the attacker exploits possible vulnerabilitites in the connections between *Valuable Machine* and *Network*, and vulnerabilities of the *Valuable Machine* itself.[6]

We note that the graph in Figure 2 consists of a couple of optimal attack paths against the infrastructure modelled in Figure 1; it is not the full graph arising from unfolding the model using the DSL specification.

C. ATTACK SIMULATIONS

An attack graph describes possible behaviour of the attacker and defender in the modelled infrastructure. The attacks, that is, ways in which the attacker can reach their target steps from initial position while respecting the AND and OR requirements of particular steps, correspond to subgraphs of the graph. While these subgraphs are not necessarily paths in the standard meaning of the word, we will call them *attack paths*.

Definition 2 (Attack Path): Let $G = (V, E, type, ttc)$ be an attack graph and let $X \subseteq A_G$. An attack graph $P = (V', E', type, ttc)$ is an *attack path in G starting in X* if

- (V', E') is a subgraph of (V, E) and $V' \subseteq A_G$,
- for every $v \in V'$, there is a node $w \in X$ for which there is a path in G starting in w and ending in v,
- if $v \in V' \setminus X$ and $type(v) = \&$, then for every parent of v in G the edge connecting v with the parent belongs to E',
- if $v \in V' \setminus X$ and $type(v) = |$, then there is a parent of v in G such that the edge connecting v with the parent belongs to E'.

Let $X_e, X_t \subseteq A_G$ be disjoint sets of attack steps, representing attack steps already executed by the attacker, and the attacker's target steps, respectively. Furthermore, let $Y \subseteq D_G$ be a set of defense steps implemented in the modelled infrastructure. We say that an attack path P in G starting in X_e *compromises* X_t *under* Y if each of the nodes in X_t belongs to the path, and none of the nodes in the path has a parent in Y. Intuitively, such a path describes an attack that starts with the attacker executing the steps in X_e, and eventually, respecting the AND/OR dependencies between the steps, leads to compromising the target attack steps, in the presence of the set of countermeasures Y.

When selecting countermeasures for increasing security of a system, one generally prefers securing the system against the attacks that are most critical in some sense. One way of determining the most critical attacks would be to extract all attacks from the model, as done for example in [23] and [8], and then compare them using a metric of choice. This approach is not feasible even in the case of models of moderate size, as the number of attacks might be exponential in the size of the model.

In this work, we use the following approach.[7] For $X_e, X_t \subseteq A_G$ and $Y \subseteq D_G$, to obtain a critical attack path in G starting in X_e and compromising X_t under Y, values are sampled from the TTC distributions assigned to the nodes. This simulates a realization of steps in the graph – now the attack steps are assigned exact, numerical values of time needed for their executions, and the defense steps are assigned a state (imple-

[6]In the DSL used here, the fact that a `Host` object is not perfectly secured is modelled by a presence of a service or software that is unknown to the modeller. This gives rise to attack steps relating to such unknown objects.

[7]It can be noted that the exact calculation of critical paths is not important for our work. It is considered part of the security definition that we aim to optimise from a black-box perspective.

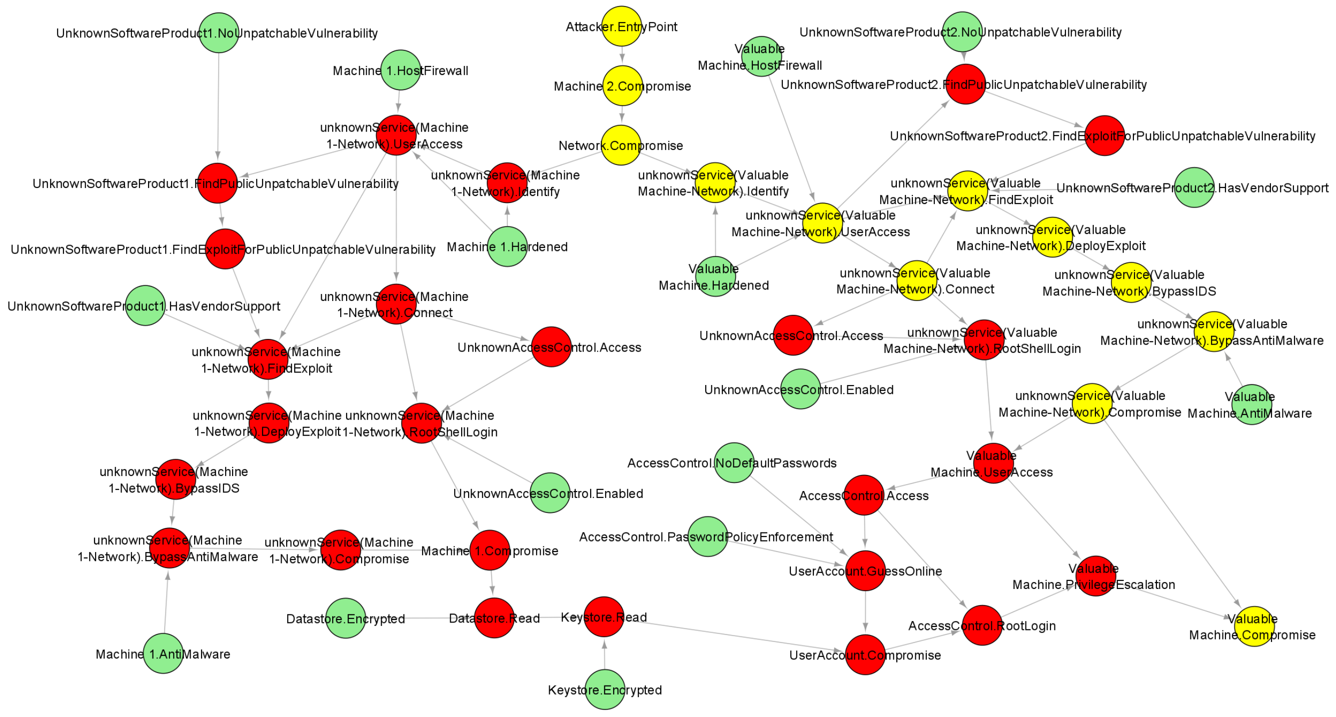
UnknownSoftwareProduct1.NoUnpatchableVulnerability
Machine 1.HostFirewall
unknownService(Machine 1-Network).UserAccess
UnknownSoftwareProduct1.FindPublicUnpatchableVulnerability
unknownService(Machine 1-Network).Identify
UnknownSoftwareProduct1.FindExploitForPublicUnpatchableVulnerability
Machine 1.Hardened
UnknownSoftwareProduct1.HasVendorSupport
unknownService(Machine 1-Network).Connect
unknownService(Machine 1-Network).FindExploit
UnknownAccessControl.Access
unknownService(Machine 1-Network).DeployExploit
unknownService(Machine 1-Network).RootShellLogin
unknownService(Machine 1-Network).BypassIDS
UnknownAccessControl.Enabled
unknownService(Machine 1-Network).BypassAntiMalware
unknownService(Machine 1-Network).Compromise
Machine 1.Compromise
Machine 1.AntiMalware
Datastore.Encrypted
Datastore.Read
Keystore.Read
Keystore.Encrypted
Attacker.EntryPoint
Machine 2.Compromise
Network.Compromise
unknownService(Valuable Machine-Network).Identify
Valuable Machine.HostFirewall
Valuable Machine.Hardened
UnknownSoftwareProduct2.NoUnpatchableVulnerability
UnknownSoftwareProduct2.FindPublicUnpatchableVulnerability
UnknownSoftwareProduct2.FindExploitForPublicUnpatchableVulnerability
unknownService(Valuable Machine-Network).UserAccess
unknownService(Valuable Machine-Network).FindExploit
UnknownSoftwareProduct2.HasVendorSupport
unknownService(Valuable Machine-Network).DeployExploit
unknownService(Valuable Machine-Network).Connect
unknownService(Valuable Machine-Network).BypassIDS
unknownService(Valuable Machine-Network).BypassAntiMalware
UnknownAccessControl.Access
unknownService(Valuable Machine-Network).RootShellLogin
UnknownAccessControl.Enabled
unknownService(Valuable Machine-Network).Compromise
Valuable Machine.AntiMalware
AccessControl.NoDefaultPasswords
Valuable Machine.UserAccess
AccessControl.Access
AccessControl.PasswordPolicyEnforcement
UserAccount.GuessOnline
Valuable Machine.PrivilegeEscalation
AccessControl.RootLogin
Valuable Machine.Compromise
UserAccount.Compromise

FIGURE 3. An attack path (in yellow) in the IT infrastructure modelled in Figure 1.

mented/not implemented). The attack path that is optimal in some sense under the sampled values is then obtained in time polynomial in the size of the graph. Repeating this process results in two lists: a list of attack paths in G starting in X_e and compromising X_t under Y, and a list of corresponding TTCs, which are *positive* numbers representing times needed for compromising the nodes in X_t when following the paths (that is, when executing the attack steps belonging to the paths, in a specific order). In the case when no attack path exists, we assume that the simulation returns an empty path with the corresponding TTC value equal to $+\infty$.

Example 3: Setting $X_e = \{$*Attacker.EntryPoint, Machine 2.Compromise*$\}$, $X_t = \{$*Valuable Machine.Compromise*$\}$ and $Y = \emptyset$ in the attack graph G from Figure 2 and using the simulations method described above, among the possible attack paths we obtain the one marked yellow in Figure 3. This attack path corresponds to the attack that is in some sense optimal from the point of view of the attacker – in at least half of the attack simulations this particular path has been returned as the one requiring the minimal amount of time. Put differently, at least half of the entries in the list of attack paths returned by the simulation method for the *Valuable Machine.Compromise* step are occupied by this particular path, and each of the entries in the bottom half of the list of TTC values corresponds to the attack following this path.

The goal of attack simulations is to assess the security of valuable assets in the modelled system and to identify ways in which they could be compromised. The objective of this work is to utilize this information for selecting countermeasures that would increase the security of the system. The suggested actions should render some of the attack paths no longer viable and in consequence increase the TTCs of the target assets.

D. REMAINING NOTATION

We now introduce the notation used in the rest of this paper.

For a list T, we use $|T|$ for the number of elements of T. If the elements of T are real numbers, then for $n \in \{1, 2, \ldots, 100\}$ we denote by T_n the *nth percentile of* T, that is, the smallest value in T such that no more than n percent of the elements of T are strictly less than the value and at least n percent of the elements of T are less than or equal to that value. In other words, T_n is the $\lceil \frac{n}{100} \cdot |T| \rceil$th element of T.

If the set of vertices of an attack graph G is not explicitly specified or not obvious from the context, we refer to it as $V(G)$. If G is an attack graph and G' and G'' are its subgraphs, i.e., $G = (V, E, type, ttc)$, $G' = (V', E', type, ttc)$ and $G'' = (V'', E'', type, ttc)$ are attack graphs satisfying $V' \subseteq V$, $V'' \subseteq V$, $E' \subseteq E$ and $E'' \subseteq E$, then the attack graph

$$G' \cup G'' := (V' \cup V'', E' \cup E'', type, ttc)$$

is the *union of* G' *and* G''.

Similarly, for two sets X, Y and relations $R \subseteq X \times X$, $S \subseteq Y \times Y$, we use $(X, R) \cup (Y, S)$ as a shorthand for $(X \cup Y, R \cup S)$.

If $(D, \preceq)$ is a partially ordered set,[8] then the set $\{x \in D\colon$ there is no $y \in D$ s.t. $x \preceq y\}$ of maximal elements in

[8]That is, $\preceq$ is a binary relation over D that is reflexive, antisymmetric and transitive.

$(D, \preceq)$ is denoted by $\max(D, \preceq)$. By $H(D, \preceq)$ we denote the set

$$\{(x, y) : x \preceq y \text{ and there is no } z \in D \text{ such that } x \preceq z \text{ and } z \preceq y\}$$

of edges of the Hasse diagram of $(D, \preceq)$.

Finally, the expected value of a random variable following a probability distribution $F \in \mathcal{F}$ is denoted by $E[F]$.

IV. PROBLEM OF SELECTING COUNTERMEASURES

The problem at hand is the following: given a model of an IT infrastructure and knowledge of possible defense steps that could be implemented, we want to suggest a set of defense steps the implementation of which increases TTC of specified attack steps, while respecting some constraints.

In Section IV-A we give a mathematical model of this problem. Due to the possibly non-deterministic nature of the underlying attack simulations method, we do not specify what an optimal solution to the problem is. Instead, for comparing the quality of candidate solutions, we use a measure defined in Section IV-B.

A. MATHEMATICAL MODELLING

We consider three elementary constraints relevant to implementation of security measures:

- limitations on available resources (e.g., fixed budget for operating costs or limited time window available for securing the system),
- order dependencies, specifying that some measures can be implemented only in a specific order (e.g., to run an antivirus scan, one needs to install an antivirus beforehand),
- mutual exclusivity between the measures (e.g., if we have two accounts for a service we might for business reasons not be allowed to disable both simultaneously).

The order dependencies are specified using the function

$$Pre : D_G \to 2^{D_G}, \tag{1}$$

which assigns to every defense step its prerequisites, i.e., the steps that *all need to* be implemented before the step itself can be implemented. Similarly, to express that execution of a step disables possibility of execution of another step, we use a function

$$Excl : D_G \to 2^{D_G}, \tag{2}$$

with $Excl(v)$ containing all the steps that cannot be implemented along v. The following examples illustrate the usage of the two functions.

Example 4: Consider again the infrastructure modelled in Figure 1. Suppose that *Datastore* is a directory and *Keystore* is its subdirectory. In such a case, to secure the *Keystore* it suffices to encrypt either the *Datastore* directory, or the *Keystore* itself; encrypting both would mean encrypting *Keystore* twice, which introduces computational overhead without increasing security. To avoid this, the modeller can set

$$Excl(Datastore.Encrypt) = \{Keystore.Encrypt\},$$
$$Excl(Keystore.Encrypt) = \{Datastore.Encrypt\}.$$

Example 5: Suppose that the *AccessControl* in the model in Figure 1 stands for login prompt when trying to access *ValuableMachine*. One way of preventing the attacker from guessing a user's password would be to force users not to use default passwords. However, should the user change their default password to, say, "password", the overall security of the system will be decreased. Thus, the modeller might require that if the default passwords are disabled, then the users are forced to create strong passwords. This could be modelled by setting

$$Pre(AccessControl.NoDefaultPasswords) = \{AccessControl.PasswordPolicyEnforcement\}.$$

We distinguish between two types of cost: monetary-like and time-like. Intuitively, computations of the overall monetary-like cost of a solution will be independent of the ordering of the defensive measures in the solution, while the overall time-like cost *will* depend on the ordering. Furthermore, we want to account for implementation of a security measure incurring possibly multiple costs of the same type. For example, hiring a security analyst requires monetary-like investments of two types: buying a computer for them (capital expenditure) and keeping them on the payroll (operating expenditure). Finally, we consider costs interdependencies. An example of such interdependency is the cost of installation of an antivirus software: the installation cost for a specific machine will depend on the number of machines for which the antivirus license has been bought.

Assuming that there are n monetary-like and m time-like resources, the costs of particular defense steps are specified using vector functions

$$\begin{aligned} Cost &: D_G \times 2^{D_G} \to \mathbb{R}^n_{\geq 0}, \\ Time &: D_G \times 2^{D_G} \to \mathbb{R}^m_{\geq 0}, \end{aligned} \tag{3}$$

with $Cost_i(v, D)$ being the ith monetary cost and $Time_j(v, D)$ being the jth time-like cost of implementation of the defense step v when v is implemented along all the defense steps in D. Making costs of steps dependent on sets of defense steps allows for encoding cost interdependencies, as illustrated by the following example.

Example 6: Suppose that the creator of the model in Figure 1 has a preferred antivirus software. Assume that the software producer offers two licenses: buying the software for one machine costs €40 per year, and for two machines the cost is €50 per year. Note that there are two defense steps corresponding to installing an antivirus software on a machine in the attack graph in Figure 2, which implies that if a solution to be implemented involves securing both machines, the yearly cost of doing so per machine will be €25. Letting $Cost_1$ represent the operating expenses, the costs of securing the machines with the antivirus could thus be expressed as

$$Cost_1(Valuable\ Machine.AntiMalware, D) = \begin{cases} 40, & \text{if } Machine\ 1.AntiMalware \notin D, \\ 25, & \text{if } Machine\ 1.AntiMalware \in D, \end{cases}$$

$$Cost_1(\textit{Machine 1.AntiMalware}, D) = \begin{cases} 40, & \text{if } \textit{Valuable Machine.AntiMalware} \notin D, \\ 25, & \text{if } \textit{Valuable Machine.AntiMalware} \in D, \end{cases}$$

for $D \in 2^{D_G}$.

The available budgets are expressed with a vector

$$\mathbf{b} = (b_1, \dots, b_n, b_{n+1}, \dots, b_{n+m}) \in \mathbb{R}^{n+m}_{\geq 0},$$

with the first n entries corresponding to the monetary costs, and the remaining ones – to the time-like costs. Intuitively, the amount of the resource i spent on implementation of a solution cannot exceed the budget b_i. Formally, we need to yet specify how this amount is computed.

Since the prerequisites function describes temporal dependencies between defense steps, we model solutions as partially ordered sets. For a solution $(D, \preceq)$ and two defense steps $w, v \in D$, the relation $w \preceq v$ means that w needs to be implemented before v can be implemented. We assume that the monetary costs are order-independent, and thus we define the cost of implementation of a solution $(D, \preceq)$ as

$$Cost_i(D, \preceq) := \sum_{v \in D} Cost_i(v, D),$$

for $i \in \{1, \dots, n\}$.

In the case of time-like resources, the total cost of the solution is computed by propagating the time costs of defense steps throughout the Hasse diagram of the solution, in a bottom-up manner. For the jth time-like resource, we use $Time_j^{(D,\preceq)}(v)$ to denote the minimal amount of the resource needed for implementation of the defense step v when implementing the solution $(D, \preceq)$, defined as

$$Time_j^{(D,\preceq)}(v) := Time_j(v, D) + \max_{\substack{v' \in D:\\ (v',v) \in H(D,\preceq)}} Time_j^{(D,\preceq)}(v').$$

Then, the amount of the resource needed for the implementation of $(D, \preceq)$, denoted by $Time_j(D, \preceq)$, is set to be the maximum among the defense steps that are not prerequisites for any other steps, i.e.,

$$Time_j(D, \preceq) := \max_{v \in \max(D,\preceq)} Time_j^{(D,\preceq)}(v).$$

The order in a solution originates from the order specified using the *Pre* function – if a defense step is a prerequisite of another one (or a prerequisite of a prerequisite, etc.), then it will precede the latter in the solution. The usage of addition and max operator in the above formulæ ensures that execution times of such dependent steps will be added to each other (the actions need to be executed in an order), and the steps that are independent in the sense of *Pre* relation will be treated as parallellizable.

The following example illustrates how the total cost of a solution is computed.

Example 7: Consider the situation when only one monetary and one time-like resource are relevant, i.e., let $n = 1$ and $m = 1$. Let

$$D_1 = \{\textit{Valuable Machine.AntiMalware}, \textit{Machine 1.AntiMalware}, \textit{AccessControl.PasswordPolicyEnforcement}, \textit{AccessControl.NoDefaultPasswords}\}$$

and

$$\preceq := \{(\textit{AccessControl.PasswordPolicyEnforcement}, \textit{AccessControl.NoDefaultPasswords})\},$$

i.e., the order $\preceq$ specifies that we need to implement *AccessControl.PasswordPolicyEnforcement* before *AccessControl.NoDefaultPasswords*, and that there are no other temporal dependencies between the defense steps belonging to D_1.

Let $Cost_1$ be defined for *Valuable Machine.AntiMalware* and *Machine 1.AntiMalware* as in Example 6. For every other defense step $v \in D_1$, set $Cost_1(v, D) = 80$ for every $D \in 2^{D_G}$. Then, we have

$$\begin{aligned} Cost_1(D_1, \preceq) &= \sum_{v \in D_1} Cost_1(v, D_1) \\ &= 25 + 25 + 80 + 80 \\ &= 210. \end{aligned}$$

Letting $Time_1(v, D) = 20$ for every $v \in D_1$ and every $D \in 2^{D_G}$, the total time needed for implementing $(D_1, \preceq)$ is computed as

$$\begin{aligned} &Time_1(D_1, \preceq) \\ &= \max\{Time_1^{(D_1,\preceq)}(\textit{Valuable Machine.AntiMalware}), \\ &\quad Time_1^{(D_1,\preceq)}(\textit{Machine 1.AntiMalware}), \\ &\quad Time_1^{(D_1,\preceq)}(\textit{AccessControl.NoDefaultPasswords})\} \\ &= \max\{20, 20, \\ &\quad Time_1(\textit{AccessControl.NoDefaultPasswords}, D_1) \\ &\quad + Time_1(\textit{AccessControl.PasswordPolicyEnforcement}, D_1)\} \\ &= \max\{20, 20, 20 + 20\} = 40. \end{aligned}$$

We are now in place to formally define the problem of selection of countermeasures. It is defined by

- Input: an attack graph G, sets $X_e, X_t \subseteq A_G$ of executed and target attack steps, functions *Pre*, *Excl*, *Cost*, *Time* satisfying (1)–(3), and a budget vector $\mathbf{b} = (b_1, \dots, b_{n+m}) \in \mathbb{R}^{n+m}_{\geq 0}$.
- Output: a partially ordered set $(D, \preceq)$, with $D \subseteq D_G$, satisfying

$$\begin{aligned} v \in D &\Rightarrow Pre(v) \subseteq D \text{ and } w \preceq v \text{ for every } w \in Pre(v), \\ v \in D &\Rightarrow Excl(v) \cap D = \emptyset, \\ Cost_i(D, \preceq) &\leq b_i, \text{ for } i \in \{1, \dots, n\}, \\ Time_j(D, \preceq) &\leq b_{n+j}, \text{ for } j \in \{1, \dots, m\}. \end{aligned} \tag{4}$$

B. QUALITY OF SOLUTIONS

The problem posed in the previous section might seem trivial: the empty partially ordered set $(\emptyset, \emptyset)$ is always a solution!

Furthermore, the specification does not mention the primary objective of a solution, which is to increase TTC of particular attack steps. This is the case, because the very term *increase of TTC* is elusive, due to the possibility of the attack simulations method being non-deterministic.

In our algorithms we will adopt the following approach. An initial run of the attack simulations will take place, resulting in TTC values that will serve as a reference point. Once candidate solutions are constructed, attack simulations will be run for each of them, and the TTCs obtained for each of the solutions will be combined with the initial values to derive an efficiency metric of the solutions. More specifically, assuming that $\mathcal{S} = \{S^{(x)}\colon x \in X_t\}$ is the set of initial lists of TTCs of target attack steps, and that $\mathcal{T} = \{T^{(x)}\colon x \in X_t\}$ is the set of lists of TTCs obtained when the implementation of a solution is simulated, we compute the efficiency score of this solution as

$$\begin{aligned}&\texttt{Efficiency}(X_t, \mathcal{S}, \mathcal{T}, c)\\ &:= \sum_{\substack{x \in X_t:\\ S_5^{(x)} \neq +\infty,\\ S_{50}^{(x)} = +\infty}} \left(1.05^{-S_5^{(x)}} \cdot \min(T_5^{(x)} - S_5^{(x)}, c)\right)\\ &\quad + \sum_{\substack{x \in X_t:\\ S_5^{(x)} \neq +\infty,\\ S_{50}^{(x)} \neq +\infty}} \Big(1.05^{-S_5^{(x)}} \cdot \min(T_5^{(x)} - S_5^{(x)}, c)\\ &\quad + 1.05^{-S_{50}^{(x)}} \cdot \min(T_{50}^{(x)} - S_{50}^{(x)}, c)\Big). \qquad (5)\end{aligned}$$

The above formula can be seen as summing up TTC improvement scores for particular target attack steps. The sum is taken over those steps for which the 5th percentile of initial TTC values is finite – if this value is not finite, then we assume that a step is initially not reachable by the attacker, which together with the assumption that implementation of a security measures cannot decrease security of the system implies that after any solution is implemented the step is still not reachable.[9]

For the steps for which the median of initial TTC values is infinite, the score computed relies on the change in the value of the 5th percentile of the TTCs. Intuitively, the greater the difference $T_5^{(x)} - S_5^{(x)}$, the higher the score of the step will be. If one was to rely solely on this difference, then increasing by the same value the TTC of a step that initially was very difficult for the attacker to execute (high initial TTC) and a step that initially was very easy for the attacker to execute (low initial TTC) would result in the same score. In other words, securing a weak part of the system and securing a strong part of the system would be scored the same. To avoid this, the factor $1.05^{-S_5^{(x)}}$ is introduced, which ensures that the score is a function decreasing in the initial TTC value. Finally, it might happen that the implementation of a solution makes the attack step no longer reachable by the attacker, or reachable in a time horizon that makes it *practically* unreachable. In this case, the value of $T_5^{(x)}$ will be either infinite (in the first case), and so the difference $T_5^{(x)} - S_5^{(x)}$ will also be infinite, or it will be large (in the second case). In both cases, the score will be equal to $1.05^{-S_5^{(x)}} c$, where c is an appropriately selected constant, depending on what is perceived by the modeller as relevant time horizon.

For the remaining steps, i.e., for the ones for which neither the 5th nor the 50th percentile of initial TTC values is infinite, the score is the sum of differences between the 5th percentiles and the medians, adjusted in the same manner as in the previous case.

[9]Recall also that the TTC values are strictly greater than zero.

V. ITERATIVE PROCEDURE FOR COUNTERMEASURES SELECTION

In this section, we describe an iterative method for selection of countermeasurse in attack graphs. The method relies on the notion of *criticality measure* of an attack step w.r.t. target steps in an attack graph. Formally, a criticality measure is a family of functions (including the non-deterministic ones)

$$Crit_G\colon A_G \times 2^{A_G} \to \mathbb{R}_{\geq 0}$$

parameterized over attack graphs. Intuitively, if for a set X_t of target nodes in an attack graph G the value of $Crit_G(v, X_t)$ is high, then the attack step v plays an important role in the attacker's reaching of the target nodes.

At each iteration of the method presented further an attempt will be made to add a set consisting of a defense step and all of its prerequisites to the solution generated so far. To refer easily to such a set, for a defense step $v \in D_G$ we denote by $\overline{Pre}(v)$ the minimal, in the sense of inclusion of both sets and relations, pair $(X, \prec)$ satisfying

1) $v \in X$, $\prec \subseteq X \times X$,
2) if $v \in X$ and $w \in Pre(v)$, then $w \in X$ and $w \prec v$.

In other words, X is the minimal set containing the defense step v, all of its prerequisites, all of the prerequisites of those prerequisites, etc., and $\prec$ is an encoding of the precedence relation.

A. OVERVIEW OF THE CORE PROCEDURE

The algorithm described in this section, Algorithm 1, constructs countermeasures suggestions as follows. At every iteration, attack simulations are performed, under the assumption that the defense steps in the solution generated so far are implemented in the system (in line 2 in the case of the initial simulations, and in line 30). The simulations return attack paths, which describe expected reasonable behaviour of the attacker, and the corresponding TTC values for the target attack steps. The attack paths are merged into a single graph, and a criticality measure is computed for every counterable attack step in this graph, in lines 5–8.

As long as there are counterable steps in the attack paths, the most critical of these steps is picked, and the defenses that counter it are examined. Once a defense step that can be added to the solution without violating any of the constraints

Algorithm 1 Simulations-Based Countermeasures Selection in Attack Graphs

```
Require: an attack graph $G$, sets $X_e, X_t \subseteq A_G$, functions $Pre$, $Excl$, $Cost$ satisfying (1)–(3), a budget vector $\mathbf{b} = (b_1, \dots, b_{n+m})$,
    a criticality measure $Crit_G$, efficiency improvement threshold $\epsilon$, efficiency constant $c$, bound $d_{iter}$ on the number of defense
    steps added to solution per iteration
Ensure: a partially ordered set $(D, \preceq)$, with $D \subseteq D_G$, satisfying (4), and its efficiency score
 1: $(D, \preceq) \leftarrow (\emptyset, \emptyset)$, $D_0 \leftarrow (\emptyset, \emptyset)$, iteration $\leftarrow 0$, partialSolutions $\leftarrow \emptyset$
 2: perform attack simulations, obtaining a set $\mathcal{S} = \{S^{(x)} : x \in X_t\}$ of lists of TTCs and a set $\mathcal{P} = \{P^{(x)} : x \in X_t\}$ of lists of
    corresponding attack paths
 3: repeat
 4:   iteration $\leftarrow$ iteration$+1$
 5:   $G' \leftarrow \bigcup_{x \in X_t} \bigcup_{P \in P^{(x)}} P$
 6:   for counterable attack step $v \in A_{G'}$ do
 7:     compute criticality score $Crit_G(v, X_t)$
 8:   end for
 9:   counterableSteps $\leftarrow$ set of counterable attack steps in $A_{G'}$
10:   defenseStepsAdded $\leftarrow 0$
11:   while counterableSteps $\neq \emptyset$ do
12:     pick a node $v \in$ counterableSteps with the highest criticality score
13:     counterableSteps $\leftarrow$ counterableSteps$\setminus\{v\}$
14:     for $w \in D_G$ such that $(w, v) \in E$ do
15:       $(D', \preceq') \leftarrow (D, \preceq) \cup \overline{Pre}(w)$
16:       if $(D', \preceq')$ is a partially ordered set, $D' \cap Excl(x) = \emptyset$ for every $x \in D'$, $Cost_i(D', \preceq') \leq b_i$ and
          $Time_j(D', \preceq') \leq b_{n+j}$ then
17:         $(D, \preceq) \leftarrow (D', \preceq')$
18:         defenseStepsAdded $\leftarrow$ defenseStepsAdded $+|D' \setminus D|$
19:         if defenseStepsAdded $\geq d_{iter}$ then
20:           go to line 30
21:         else
22:           remove defense steps in $\overline{Pre}(w)$ from $G'$
23:           remove attack steps countered by defense steps in $\overline{Pre}(w)$ from counterableSteps and from $G'$
24:           recompute criticality scores of attack steps in counterableSteps
25:           go to line 12
26:         end if
27:       end if
28:     end for
29:   end while
30:   perform attack simulations in the model with all the defense steps in $D$ implemented, obtaining a set
      $\mathcal{T} = \{T^{(x)} : x \in X_t\}$ of lists of TTCs and a set $\mathcal{P} = \{P^{(x)} : x \in X_t\}$ of lists of corresponding attack paths
31:   $D_{\text{iteration}} \leftarrow (D, \preceq)$, $E_{\text{iteration}} \leftarrow \texttt{Efficiency}(X_t, \mathcal{S}, \mathcal{T}, c)$
32:   partialSolutions $\leftarrow$ partialSolutions $\cup\{(D_{\text{iteration}}, E_{\text{iteration}})\}$
33: until a stop condition is satisfied or $D_{\text{iteration}} = D_{\text{iteration}-1}$
34: $k \leftarrow$ minimal $i$ such that $(D_i, E_i) \in$ partialSolutions and $\left|\frac{E_{\text{iteration}} - E_i}{E_{\text{iteration}}}\right| < \epsilon$
35: return $D_k$ and $E_k$
```

is encountered, this defense step and all of its prerequisites are added to the solution, in line 17, and the set of counterable steps is updated. Analysis of the counterable attack steps stops once all of them have been examined or when the number of defense steps added to the solution in the current iteration exceeds the prespecified bound d_{iter}. Then, the next iteration starts. The main loop of the algorithm stops once a stop condition is met (e.g., a desired improvement in TTC values has been achieved) or an iteration passed without altering the solution (in line 33).

The role of the d_{iter} parameter is to enable a trade-off between the quality of the solution produced and the number of simulations performed. To obtain a high quality solution, one could run a very high number of iterations of the main loop of the algorithm, adding to the solution exactly one defense step per iteration. This can be seen as repeated identification of the most critical threats and implementation of the smallest possible change in the system improving the security against these threats (a change that, given the threats identified, seems to be optimal in some sense). However, while simulations are an effective way of identifying some of the optimal attack paths even in large attack graphs, they remain time–consuming. In the case of big models, it might be preferable to use as the stop condition a limit on the number of iterations (being at the same time a limit on the number of performed simulations). Since new elements are added to the solution per iteration, such a limit indirectly limits the size of the solution, possibly reducing its effectiveness. To avoid producing low quality solutions when a limit on the number of simulations is small, one can increase the value of d_{iter}, ensuring that, if possible, at least d_{iter} defense steps will be added to the solution in each iteration.

To avoid overinvesting into measures that increase security only marginally, we keep track of the results of particular iterations. After each iteration, the efficiency of the solution generated so far is computed, using formula (5), and both the solution and the metric are stored (in lines 31-32). Once the main loop executes, the efficiencies of partial results are compared against the efficiency of the final solution. This is done by computing the relative improvement of the final solution w.r.t. each of the partial solutions, in line 34.[10] The smallest, in the sense of number of defense step it contains, partial solution for which the relative improvement does not exceed some specified threshold ϵ is eventually returned, along with its efficiency, in line 35.

B. MEASURING CRITICALITY OF ATTACK STEPS

At the heart of the iterative procedure described in the previous section lies the idea of ranking counterable attack steps by their criticality. We consider several basic criticality measures that can be used for deriving criticality scores of attack steps.

1) CRITICALITY MEASURES

Let $G = (V, E, type, ttc)$ be an attack graph and let $\mathcal{P} = \{P^{(x)} : x \in X_t\}$ be a set of lists of attack paths in G compromising target attack steps in $X_t \subseteq A_G$ under $Y \subseteq D_G$. Set $G' := \bigcup_{x \in X_t} \bigcup_{P \in P^{(x)}} P$. For $v \in A_{G'}$, let

$$freq(v) := \sum_{x \in X_t} |\{P \in P^{(x)} : v \in V(P)\}|, \tag{6}$$

$$out_w(v) := \sum_{(v,v') \in E(G')} freq(v'), \tag{7}$$

$$out_w^{AND}(v) := \sum_{(v,v') \in E(G'), type(v')=AND} freq(v'), \tag{8}$$

$$dist_e(v) := 1/dist_{G'}(X_e, Y, v). \tag{9}$$

The *frequency* function *freq* defined by (6) returns the number of attack paths in which the attack step v appears. In other words, it counts how frequently the attacker used this step in simulations. If this value is high, then the attack step contributes to many possible attacks on target assets, and so the step can be deemed critical.

[10]Note that in line 34, the value of $E_{\text{iteration}}$ is equal to the efficiency of the solution obtained in the very last iteration of the main loop.

The function defined by (7) is the *weighted outdegree* of v in G'. Intuitively, the more attack steps are enabled by v, the more critical v is. However, v will be more critical if its children attack steps have high frequencies. Thus, instead of simply counting the children, the sum of their frequencies is computed.

Observe that to render an AND attack step useless it is enough to disable one of its parents. In other words, disabling a parent of an AND attack step indirectly disables also that step. Thus, from the defender's point of view, it might be more worthwile to implement a defense that disables the parent rather than the AND child itself. Combining this intuition with the one from the previous paragraph, we arrive at the *weighted outdegree over AND children* criticality metric, defined by (8).

Finally, there might be an incentive to stop the attacker from penetrating the system as soon as possible. In such a case, the attack steps considered to be most critical will be the ones lying "the closest" to the attack steps compromised so far. Various metrics could be applied as the $dist_{G'}$ function, but the selected one should take into account the time needed for execution of particular attack steps as well as the AND/OR structure of the attack graphs. We give one example of such a metric in Appendix A.

2) CRITICALITY SCORES

The criticality measures described in the previous section can be directly used as the criticality scores in Algorithm 1. For a finer distinction between the criticality of attack steps, a number of these measures could be taken into account simultaneously.

One way of doing this is as so: select any number of criticality measures and order them by their perceived importance. For every counterable attack step, create a vector of the measures, with the vector entries ordered by the importance. Then, sort the vectors using a variant of the lexicographical order, thus sorting the steps from the most to the least critical one.

Example 8: Frequencies and weighted outdegrees of counterable attack steps in the attack graph from Figure 2 are given in Table 1. The steps in the table are sorted first w.r.t. the frequency, and then w.r.t. the weighted outdegree.

The criticality scores are computed here as follows. The most critical step (i.e., the one in the top row of the table) receives criticality score equal to the number of counterable attack steps. Then, we iterate over the remaining steps, from the most critical to the least critical one. For every step, its criticality score will be equal to the score of the previous one if their criticality metrics are the same. Otherwise, its score will be smaller by one.

VI. VARIANTS OF ALGORITHM 1

A. FOCUSING ON ATTACK STEPS WITH ADDITIONAL PROPERTIES

The process of selection of attack steps to be countered in Algorithm 1 is driven by the chosen criticality score. While

TABLE 1. Selected criticality measures of counterable attack steps in attack graph from Figure 2.

Attack step	Frequency	Weighted outdegree	Criticality score
unknownService(Valuable Machine-Network).UserAccess	8	16	15
unknownService(Valuable Machine-Network).Identify	8	8	14
unknownService(Valuable Machine-Network).FindExploit	6	6	13
unknownService(Valuable Machine-Network).BypassAntiMalware	6	6	13
UserAccount.GuessOnline	3	6	12
unknownService(Machine 1-Network).UserAccess	3	6	12
Keystore.Read	3	6	12
unknownService(Machine 1-Network).Identify	3	3	11
Datastore.Read	3	3	11
unknownService(Valuable Machine-Network).RootShellLogin	2	6	10
UnknownSoftwareProduct2.FindPublicUnpatchableVulnerability	2	2	9
unknownService(Machine 1-Network).FindExploit	2	2	9
unknownService(Machine 1-Network).BypassAntiMalware	2	2	9
unknownService(Machine 1-Network).RootShellLogin	1	3	8
UnknownSoftwareProduct1.FindPublicUnpatchableVulnerability	1	1	7

this is reasonable, in some cases it might be insufficient for creating solutions suited for a given situation. For instance, if the modelled system is currently under attack, it might be wasteful to implement a countermeasure that prevents the attacker from performing an attack step that they are expected to reach in 100 days. In such a case, the analysis of counterable attack steps should be narrowed to these steps that, given the current position of the attacker, are likely to be executed in the near future.

More generally, the algorithm could be adapted so that the analysis focuses only on the subset of currently counterable attack steps (selected in line 11) that satisfy additional properties.

Example 9: Assume that there is only one time budget, b_{n+1}, and that the remaining budgets are monetary-like. Suppose that the system is under attack and let $Exp_G(X_e, Y, v)$ denote the expected time needed by the attacker to execute all of the prerequisites of the attack step v,[11] with $Y \subseteq D_G$ being currently implemented countermeasures. To stop the attacker from progressing through the system as soon as possible, one could introduce parameter t_u, which together with the time budget would specify the time window $[b_{n+1}, t_u]$. Replacing line 11 of Algorithm 1 with

$$\text{counterableSteps} \leftarrow \text{set of counterable attack steps } v \text{ in } A_{G'} \text{ satisfying } Exp_G(X_e, Y, v) \in [b_{n+1}, t_u]$$

will result in the algorithm considering only those attack steps that the attacker is expected to start executing, approximately, in a point in time belonging to this time window, and the countermeasures against which could be implemented within the specified time budget.

B. SELECTION BASED ON QUALITY OF DEFENSE STEPS

In each iteration, Algorithm 1 focuses on countering *the most critical attack steps*. Alternatively, a measure of quality of defense steps could be used, and the focus could be put on selecting a (set of) defense step(s) of the highest quality (as it is done, e.g., in [9], [33], and [23]). In the context of our framework, it is important to remember that at each iteration of Algorithm 1 only partial knowledge about the attacks against the modelled infrastructure is available. In particular, while it is known which of the attack steps lie on *the currently optimal attack paths*, the exact importance of the remaining attack steps is not known. In consequence, measuring quality of defense steps countering only the latter attack steps is not straightforward. Since some of these defense steps might be prerequisites for the defenses countering attack steps that do lie on optimal attack paths, the overall gain from adding them to the solution is difficult to assess.

Let $G, X_t, \mathcal{P}$ and G' be as defined in the first paragraph of Section V-B1. For a defense step $w \in D_{G'}$, we denote by $\texttt{Counter}(w)$ the set

$$\{v \in A_{G'} : (w', v) \in E(G') \text{ for some } w' \in \overline{Pre}(w)\}$$

of all attacks steps in the optimal attack paths that are counterable by a defense step in $\overline{Pre}(w)$. Among the ways of measuring the quality of w are

$$q_1(w) := |\texttt{Counter}(w)|, \tag{10}$$

$$q_2(w, Crit_G) := \sum_{v \in \texttt{Counter}(w)} Crit_G(v, X_t), \tag{11}$$

$$q_3(w, Crit_G) := \max_{v \in \texttt{Counter}(w)} Crit_G(v, X_t), \tag{12}$$

$$q_4(w, Crit_G) := \frac{q_2(w, Crit_G)}{q_1(w)}. \tag{13}$$

The quality measure (10) simply counts the number of attack steps countered by w and its prerequisites in the optimal attack paths. The metrics defined by (11) and (12) compute, respectively, the sum and the maximum of criticality scores of these attack steps. The last of the four metrics is the average criticality of attack steps countered by the defense steps in $\overline{Pre}(w)$. Each of these measures could be modified further, for instance, with division by the cost of implementation of $\overline{Pre}(w)$ w.r.t. to a specific resource, or by the average cost over all resources. This would result in a metric assessing the average improvement in system's security per unit of resource invested in implementing $\overline{Pre}(w)$.

[11] One possible way of computing this value is given in Appendix IX.

TABLE 2. Selected parameters for Algorithm 1 used in particular experiments.

Experiment	Model	Maximal number of iterations	$\mathbf{d_{iter}}$	Criticality metric
U5f	Ukraine	5	4	f
U5o	Ukraine	5	4	out_w
U5fo	Ukraine	5	4	(f, out_w)
U15f	Ukraine	15	1	f
U15o	Ukraine	15	1	out_w
U15fo	Ukraine	15	1	(f, out_w)
Sf	SEGRID	5	4	f
So	SEGRID	5	4	out_w
Sfo	SEGRID	5	4	(f, out_w)

Should additional information be available to the modeller, such as expected annual loss incurred by a countermeasure being not implemented in the system, security-oriented variants of the return on investment (ROI) metric, like the ones considered in [9], [31], and [43], could also be employed for measuring quality of defenses.

VII. EMPIRICAL VALIDATION

To validate the framework described in the previous sections in practice, we have implemented it in a prototype tool. As the backend for generating attack graphs and performing attack simulations, we used the commercial tool securiCAD [7]. The remaining components of Algorithm 1 are implemented as a Python script.

We have tested the framework on two realistic models that we will refer to as the *Ukraine model*[12] and the *SEGRID model*.[13] The former consists of 106 objects, and the latter contains 728 objects. In the DSL used, there are on average 5.8 steps related to an object, hence the attack graphs corresponding to the two models are expected to have approximately 610 and 4220 nodes, respectively. For the Ukraine model, synthetic constraints have been generated, including the *Excl* and *Pre* functions, as well as interdependent costs of implementation of defense measures. No constraints were used in the case of the SEGRID model.

Several experiments have been performed on the two models. In each of them, the efficiency improvement threshold ϵ has been set to 0.1, the efficiency constant c to 150 ,[14] and the number of simulations performed by securiCAD was set to 100. The defense step selection rule employed was the one described in Section VI-B, and the quality metric used was the metric q_3 given by (12). Three variants of the metric were used, using as the criticality score for attack steps the *frequency*, given by (6), *weighted outdegree*, given by (7), and the combination of the two described in Section V-B2 and illustrated in Example 8. In the experiments the *frequency* parameter was limited to looking at the frequency of the nine most common attack paths (due to a technical limitation in the securiCAD tool.) The stop condition used was a limit on number of executions of the main loop (lines 4-32) of the algorithm, with earlier stop happening only if the fifth percentiles of TTC values of all target attack steps attained the value of $+\infty$, i.e., if $T_5^{(x)} = +\infty$ for every $x \in X_t$. A summary of parameters selected for particular experiments is presented in Table 2. In each of these experiments, there were three target attack steps.

The Python scripts have been run on a Windows machine running Intel Core i7-8665U CPU at 1.9 and 2.11 GHz dual core, with 16 GB of RAM, and the securiCAD instance employed have been run on a Google Cloud virtual machine running Ubuntu 16.04, with two virtual CPUs and 7.5 GB of RAM.

The main objective of the tests was to verify that our framework is usable in practice. To this end, we have measured the execution times of the experiments and the quality of the produced solutions. An extract from the obtained results is presented in Figure 4, Figure 5 and Figure 6.

Figure 4 illustrates the impact of the size of the model and the selected criticality metric on the running time. Greater variance in running times has been observed in the case of the bigger model. While the SEGRID model contains almost seven times more objects than the Ukraine model, the medians of running times for both models are similar – for the former, they oscilate around 32 minutes, and around 28 minutes for the latter. This suggests that our approach scales well with the size of the model.

In Figure 5 the computation times for the U5f experiment with varying number of target attack steps are plotted. It is expected that the time will grow with the number of target steps, since the greater the number, the more attack paths are used for creating the graph in line 5 of Algorithm 1, and the bigger the graph itself. The median of sizes of the graph obtained for particular number of target steps are given in top part of the figure. Starting from four target steps,

[12]Model files available at `https://github.com/mal-lang/securicad-coa-generator/tree/master/paper_tests_ukraine/ukraineTestModel`

[13]Model files available at `https://segrid.eu/wp-content/uploads/2018/01/uc2-sc2_sCADfiles-1.zip`.
See [38] for a detailed description of the model.

[14]The time unit used by securiCAD is day, and so by setting $c = 150$ we deem the 150 days increase of the 5th and the 50th percentiles of TTC values of an attack step to be fully satisfying. Further TTC increase will have no impact on the efficiency score.

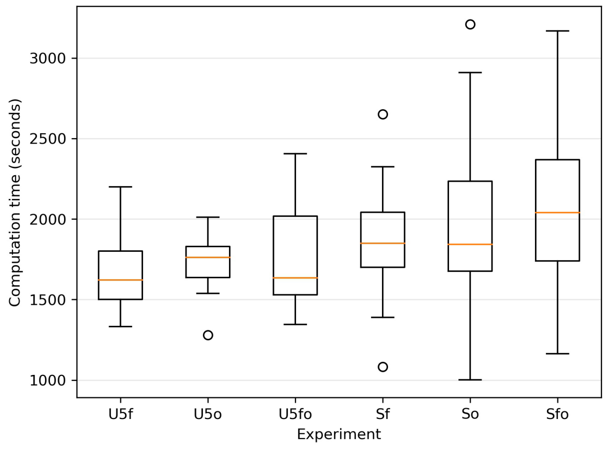

FIGURE 4. **Boxplots of computation times. For each experiment setup, 20 runs have been performed. Orange bars correspond to medians, and the lower and the upper borders of the boxes to first and third quartiles, respectively. Circles represent outliers.**

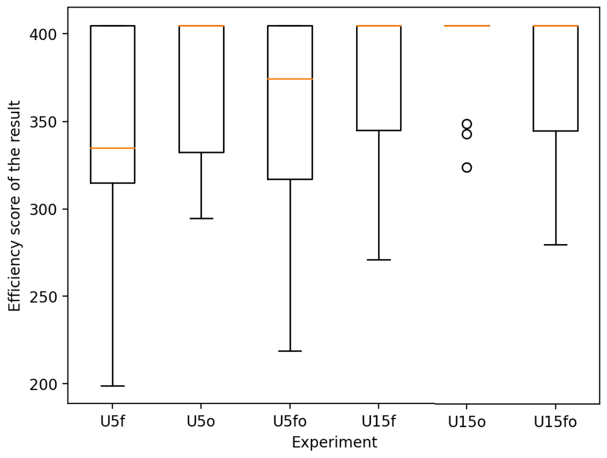

FIGURE 6. **Boxplots of efficiency scores. For each experiment setup, 20 runs have been performed. Orange bars correspond to medians, and the lower and the upper borders of the boxes to first and third quartiles, respectively. Circles represent outliers.**

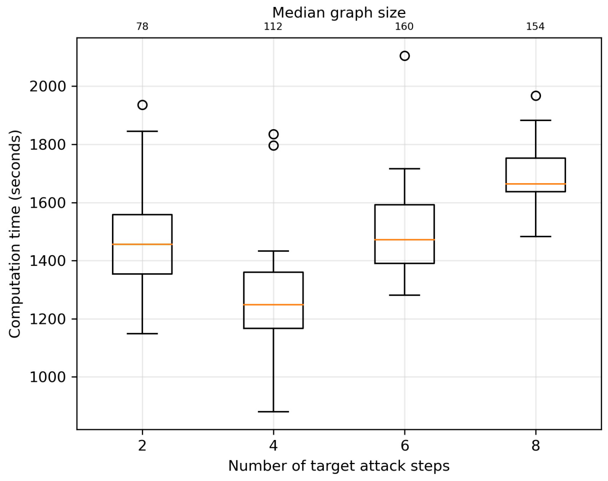

FIGURE 5. **Boxplots of computation times for U5f experiment. For each experiment setup, 20 runs have been performed. Orange bars correspond to medians, and the lower and the upper borders of the boxes to first and third quartiles, respectively. Circles represent outliers.**

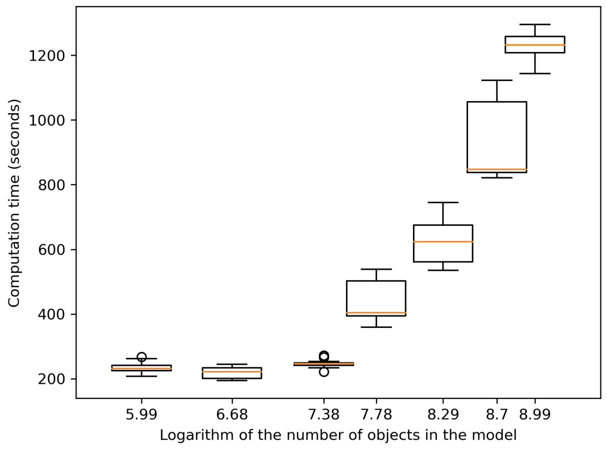

FIGURE 7. **Boxplots of computation times for models of different sizes. For each model in the scalability experiment, 20 runs have been performed. Orange bars correspond to medians, and the lower and the upper borders of the boxes to first and third quartiles, respectively. Circles represent outliers.**

an increase in both minimal and maximal computation times can be observed.

Differences in quality of produced results are illustrated in Figure 6. Both in the case of U5x and U15x experiments, employing the weighted outdegree criticality metric resulted in smaller range of efficiency scores of produced results. Additionally, the median of efficiency scores obtained in U5o is higher than the medians of U5f and U5fo. The same tendency has been observed also in the case of the SEGRID model. This suggests that the weighted outdegree criticality metric might be preferable to the other two metrics when the defense steps selection rule employed here is used.

Not surprisingly, results of higher quality are produced when more iterations of the algorithm are executed, and when less defense steps are selected per iteration – the ranges of values obtained in U15x experiments are smaller than those obtained in U5x experiments, and the medians of the former are higher.

To investigate further the scalability of our framework we have created synthetic models of different sizes (having from 400 up to 8000 objects)[15] producing different sized attack graphs. The corresponding running times are plotted in Figure 7, using logarithmic scale for better visibility. It can be seen that even for large models the proposed solution is producing results within a useful and practical time duration, and that the running time grows linearly with the number of objects in the model.

[15]Model files available at `https://github.com/mal-lang/securicad-coa-generator/tree/master/models-for-scalability-test`

For the scalability tests the latest version of the Python scripts have been run on a Windows machine running Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz 1.90 GHz, with 16 GB of RAM, and the securiCAD instance employed have been run on a Google Cloud virtual machine running Ubuntu 16.04, with eight virtual CPUs and 16 GB of RAM.

VIII. CONCLUSION

The main goal of the work presented in this paper was to devise a method for suggesting efficient security countermeasures in IT infrastructures modelled using MAL. We proposed a scalable and highly tunable algorithm that constructs sets of countermeasures by repeated analysis of optimal attack paths in the modelled system. By testing our framework on realistic models using a prototype implementation, we demonstrated that it is usable in practice. Furthermore, we have gained some insights into the impact of several criticality measures on the quality of the produced results.

Applicability of our methodology relies on the existence of a DSL suitable for modelling the system of interest. We could use any of the existing MAL based DSLs depending on the domain of application.

The main motivation for developing the algorithm has been to support security engineers with a capability to suggest actionable design decisions given a certain ICT infrastructure, without having to engage in a trial and error process of design alternatives. Primarily, this saves time for the security engineer. Our tests indicate that the algorithm seem to deliver results with decent performance. In a scenario where the security engineer is working with strategic or tactic design decisions, the performance is probably sufficient. However, in the case where design decisions are taken as a response to an ongoing attack, calculations could possibly take too long. Even though the algorithm can be fed with a time budget representing the available window to react, the algorithm does not take its own computing time into consideration. With empirical data on computational performance for a given model this time could be introduced as an additional cost parameter in the algorithm in a future extension.

In this work, we have not considered optimization w.r.t. the budget of the defender. A natural, yet not very efficient, approach towards striking a balance between the defender's investment and efficiency of solutions would be to simply run Algorithm 1 for different values of budgets and compare the results. Possible future work includes developing more efficient ways of tackling this problem.

A shortcoming with the prototype is that it assumes a static attack graph. In practice, activating defense steps is not the only design alternative available to the security engineer, introducing changes that mean structural changes in the DSL instance model is also an option. This however also means that the resulting attack graph changes. Here we can thus note that the practical usefulness of the prototype will vary with the design of the DSL. In cases where the DSL is encoding defense mechanisms to a large extent as defense steps the algorithm would be more relevant than in cases where defenses are captured as assets and asset relations. For instance, we could imagine one DSL having a defense step representing two-factor authentication protection while another would feature credentials as separate assets with a relation to the protected software asset. Ongoing work includes adapting the prototype presented here to also encompass structural changes.[16]

Although Algorithm 1 has been tailored to fit the MAL-generated attack graphs, we note that it is somewhat universal. In its formulation, one could replace attack graphs with another graphical model for security, attack paths with appropriate notion of an attack, and attack simulations with a corresponding method of determining optimal attacks in a model. The resulting algorithm would produce countermeasures suggestions in such alternative framework. For instance, the algorithm could be immediately adapted for attack-defense trees [21] in which basic actions are assigned probability distributions of time execution (as in [3]), with attacks being elements of the *set semantics* [4] of a tree, and with the method described in Section 4.5 of [41] used for determining an optimal attack. It seems that this method could be further adapted to fit attributes other than TTC, e.g., attacker's investment.

IX. APPENDIX A COMPUTING DISTANCE IN ATTACK GRAPHS

To measure how far is the attacker from executing an attack step in an attack graph, one can compute the expected time needed by the attacker for executing all of the prerequisites of the step. This measure can be formalized as follows. Let $G = (V, E, type, ttc)$ be an attack graph, $X_e \subseteq A_G$ be a set of attack steps executed so far by the attacker, and $Y \subseteq D_G$ be a set of defences implemented in G. An attack step $v \in A_G$ is *reachable by the attacker from* X_e *under* Y if there is an attack path in G starting in X_e and compromising $\{v\}$ under Y.

For $v \in A_G$, let $Exp_G(X_e, Y, v)$ denote the expected time needed by the attacker to execute all of the prerequisites of the attack step v when starting in X_e and when the defenses in Y are implemented in the system. If v is not reachable from X_e under Y, set $Exp_G(X_e, Y, v) = +\infty$. For a reachable attack step v, this value is defined as

$$Exp_G(X_e, Y, v) := \begin{cases} 0, \text{ if } v \in X_e, \\ \min\limits_{v' :\, (v',v) \in E} Exp_G(X_e, Y, v') + E[ttc(v')], \\ \qquad\qquad \text{if } v \notin X_e \text{ and } type(v) = |, \\ \max\limits_{v' :\, (v',v) \in E} Exp_G(X_e, Y, v') + E[ttc(v')], \\ \qquad\qquad \text{if } v \notin X_e \text{ and } type(v) = \&. \end{cases}$$

[16]Of course, if MAL featured some other underlying formalism for analyzing the attack graphs, such as a game theoretic calculation engine, also other approaches, with other properties, would be possible to develop. Such work, however, has been considered out of scope.

REFERENCES

[1] M. Albanese and S. Jajodia, ''A graphical model to assess the impact of multi-step attacks,'' *J. Defense Model. Simul., Appl., Methodol., Technol.*, vol. 15, no. 1, pp. 79–93, Jan. 2018.

[2] P. Ammann, D. Wijesekera, and S. Kaushik, ''Scalable, graph-based network vulnerability analysis,'' in *Proc. 9th ACM Conf. Comput. Commun. Secur. (CCS)*, Washington, DC, USA, 2002, pp. 217–224.

[3] F. Arnold, H. Hermanns, R. Pulungan, and M. Stoelinga, ''Time-dependent analysis of attacks,'' in *Principles of Security and Trust*. Grenoble, France: Springer, 2014, pp. 285–305.

[4] A. Bossuat and B. Kordy, ''Evil twins: Handling repetitions in attack-defense trees—A survival guide,'' in *Graphical Models for Security—4th International Workshop, GraMSec 2017*. Santa Barbara, CA, USA: Springer, Aug. 2017, pp. 17–37.

[5] G. Brown, M. Carlyle, J. Salmerón, and K. Wood, ''Defending critical infrastructure,'' *Interfaces*, vol. 36, no. 6, pp. 530–544, Nov. 2006.

[6] K. Edge, G. Dalton, R. Raines, and R. Mills, ''Using attack and protection trees to analyze threats and defenses to homeland security,'' in *Proc. MILCOM*, Oct. 2006, pp. 1–7.

[7] M. Ekstedt, P. Johnson, R. Lagerström, D. Gorton, J. Nydren, and K. Shahzad, ''SecuriCAD by Foreseeti: A CAD tool for enterprise cyber security management,'' in *Proc. 19th IEEE Int. Enterprise Distrib. Object Comput. Workshop*, Adelaide, SA, Australia, Sep. 2015, pp. 152–155, 2015.

[8] B. Fila and W. Widel, ''Exploiting attack–defense trees to find an optimal set of countermeasures,'' in *Proc. IEEE 33rd Comput. Secur. Found. Symp. (CSF)*, Boston, MA, USA, Jun. 2020, pp. 395–410.

[9] G. G. Granadillo, M. Belhaouane, H. Debar, and G. Jacob, ''RORI-based countermeasure selection using the OrBAC formalism,'' *Int. J. Inf. Secur.*, vol. 13, no. 1, pp. 63–79, Feb. 2014.

[10] S. Hacks, S. Katsikeas, E. Ling, R. Lagerström, and M. Ekstedt, ''PowerLang: A probabilistic attack simulation language for the power domain,'' *Energy Informat.*, vol. 3, no. 1, pp. 1–17, Dec. 2020.

[11] R. R. Hansen, K. G. Larsen, A. Legay, P. G. Jensen, and D. B. Poulsen, ''ADTLang: A programming language approach to attack defense trees,'' *Int. J. Softw. Tools Technol. Transf.*, vol. 23, no. 1, pp. 89–104, Feb. 2021.

[12] H. Holm, M. Ekstedt, and D. Andersson, ''Empirical analysis of system-level vulnerability metrics through actual attacks,'' *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 6, pp. 825–837, Nov./Dec. 2012.

[13] H. Holm and T. Sommestad, ''So long, and thanks for only using readily available scripts,'' *Inf. Comput. Secur.*, vol. 25, no. 1, pp. 47–61, 2017.

[14] N. Idika and B. Bhargava, ''Extending attack graph-based security metrics and aggregating their application,'' *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 1, pp. 75–85, Jan. 2012.

[15] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer, ''Modeling modern network attacks and countermeasures using attack graphs,'' in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2009, pp. 117–126.

[16] P. Johnson, R. Lagerström, and M. Ekstedt, ''A meta language for threat modeling and attack simulations,'' in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, New York, NY, USA, Aug. 2018, pp. 1–8.

[17] S. Katsikeas, S. Hacks, P. Johnson, M. Ekstedt, R. Lagerström, J. Jacobsson, M. Wällstedt, and P. Eliasson, ''A probabilistic attack simulation language for the IT domain,'' in *Proc. Graph. Models Secur. 7th Int. Workshop (GraMSec)*, 2020, pp. 67–86.

[18] S. Katsikeas, P. Johnson, S. Hacks, and R. Lagerström, ''Probabilistic modeling and simulation of vehicular cyber attacks: An application of the meta attack language,'' in *Proc. 5th Int. Conf. Inf. Syst. Secur. Privacy*, Prague, Czech Republic, 2019, pp. 175–182.

[19] S. Katsikeas, P. Johnsson, S. Hacks, and R. Lagerström, ''VehicleLang: A probabilistic modeling and simulation language for modern vehicle IT infrastructures,'' *Comput. Secur.*, vol. 117, Jun. 2022, Art. no. 102705.

[20] M. Khouzani, Z. Liu, and P. Malacaria, ''Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs,'' *Eur. J. Oper. Res.*, vol. 278, no. 3, pp. 894–903, Nov. 2019.

[21] B. Kordy, S. Mauw, S. Radomirovic, and P. Schweitzer, ''Attack-defense trees,'' *J. Log. Comput.*, vol. 24, no. 1, pp. 55–87, Feb. 2014.

[22] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, ''DAG-based attack and defense modeling: Don't miss the forest for the attack trees,'' *Comput. Sci. Rev.*, vols. 13–14, pp. 1–38, Nov. 2014.

[23] F. Li, Y. Li, S. Leng, Y. Guo, K. Geng, Z. Wang, and L. Fang, ''Dynamic countermeasures selection for multi-path attacks,'' *Comput. Secur.*, vol. 97, Oct. 2020, Art. no. 101927.

[24] P. Mukherjee and C. Mazumdar, ''Attack difficulty metric for assessment of network security,'' in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, Aug. 2018, p. 44.

[25] P. Nespoli, D. Papamartzivanos, F. G. Mármol, and G. Kambourakis, ''Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks,'' *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1361–1396, 2nd Quart., 2018.

[26] X. Ou, S. Govindavajhala, and A. W. Appel, ''MulVAL: A logic-based network security analyzer,'' in *Proc. USENIX Secur.*, 2005, pp. 113–128.

[27] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup, ''A weakest-adversary security metric for network configuration security analysis,'' in *Proc. 2nd ACM Workshop Quality Protection (QoP)*, Alexandria, VA, USA, 2006, pp. 31–38.

[28] C. Phillips and L. P. Swiler, ''A graph-based system for network-vulnerability analysis,'' in *Proc. Workshop New Secur. Paradigms (NSPW)*, Charlottsville, VA, USA, 1998, pp. 71–79.

[29] N. Poolsappasit, R. Dewri, and I. Ray, ''Dynamic security risk management using Bayesian attack graphs,'' *IEEE Trans. Dependable Secur. Comput.*, vol. 9, no. 1, pp. 61–74, Jan. 2012.

[30] A. Roy, D. S. Kim, and K. S. Trivedi, ''Attack countermeasure trees (ACT): Towards unifying the constructs of attack and defense trees,'' *Secur. Commun. Netw.*, vol. 5, no. 8, pp. 929–943, 2012.

[31] A. Roy, D. S. Kim, and K. S. Trivedi, ''Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees,'' in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Boston, MA, USA, Jun. 2012, pp. 1–12.

[32] T. Sawik, ''Selection of optimal countermeasure portfolio in IT security planning,'' *Decis. Support Syst.*, vol. 55, no. 1, pp. 156–164, Apr. 2013.

[33] A. Shameli-Sendi, H. Louafi, W. He, and M. Cheriet, ''Dynamic optimal countermeasure selection for intrusion response system,'' *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 755–770, Sep. 2018.

[34] O. Sheyner, J. W. Haines, S. Jha, R. Lippmann, and J. M. Wing, ''Automated generation and analysis of attack graphs,'' in *Proc. IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, May 2002, pp. 273–284.

[35] J. Soikkeli, L. Muñoz-González, and E. Lupu, ''Efficient attack countermeasure selection accounting for recovery and action costs,'' in *Proc. 14th Int. Conf. Availability, Rel. Secur.*, Canterbury, U.K., Aug. 2019, p. 3.

[36] T. Sommestad and F. Sandström, ''An empirical test of the accuracy of an attack graph analysis tool,'' *Inf. Comput. Secur.*, vol. 23, no. 5, pp. 516–531, Nov. 2015.

[37] O. Stan, R. Bitton, M. Ezrets, M. Dadon, M. Inokuchi, Y. Ohta, T. Yagyu, Y. Elovici, and A. Shabtai, ''Heuristic approach towards countermeasure selection using attack graphs,'' 2019, *arXiv:1906.10943*.

[38] A. Vernotte, M. Välja, F. Fransen, M. Ekstedt, and G. Björkman, ''SEGRID detailed reference model use case 2 scenario 2,'' White Paper, 2018.

[39] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, ''An attack graph-based probabilistic security metric,'' in *Proc. 22nd Annu. IFIP WG Working Conf. Data Appl. Secur.* London, U.K.: Springer, 2008, pp. 283–296.

[40] L. Wang, A. Singhal, and S. Jajodia, ''Toward measuring network security using attack graphs,'' in *Proc. ACM Workshop Quality Protection (QoP)*, Alexandria, VA, USA, 2007, pp. 49–54.

[41] W. Wideł, ''Formal modeling and quantitative analysis of security using attack-defense trees. (Modélisation formelle et analyse quantitative de la sécurité à l'aide d'arbres les attaques et de défense),'' Ph.D. thesis, IRISA, INSA Rennes, France, 2019.

[42] W. Wideł, M. Audinot, B. Fila, and S. Pinchinat, ''Beyond 2014: Formal methods for attack tree–based security modeling,'' *ACM Comput. Surv.*, vol. 52, no. 4, p. 75, Aug. 2019.

[43] T. Yaqoob, A. Arshad, H. Abbas, M. F. Amjad, and N. Shafqat, ''Framework for calculating return on security investment (ROSI) for security-oriented organizations,'' *Future Gener. Comput. Syst.*, vol. 95, pp. 754–763, Jun. 2019.

[44] J. Zeng, S. Wu, Y. Chen, R. Zeng, and C. Wu, ''Survey of attack graph analysis methods from the perspective of data and knowledge processing,'' *Secur. Commun. Netw.*, vol. 2019, pp. 1–16, Dec. 2019.

[45] K. Zheng, L. A. Albert, J. R. Luedtke, and E. Towle, ''A budgeted maximum multiple coverage model for cybersecurity planning and management,'' *IISE Trans.*, vol. 51, no. 12, pp. 1303–1317, Dec. 2019.

WOJCIECH WIDEŁ received the Ph.D. degree in mathematics from the AGH University of Science and Technology, Kraków, Poland, in 2017, and the second Ph.D. 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.

PREETAM MUKHERJEE received the M.Eng. and Ph.D. degrees in computer science from Jadavpur University, Kolkata, India, in 2011 and 2020, respectively. He is currently working as an Assistant Professor of cyber security with Digital University Kerala, India. He was a Postdoctoral Researcher with the Division of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden. He has previously worked at IIT Kharagpur, India, and at the Centre for Distributed Computing, Jadavpur University. His research interests include security analysis, business process modeling, and security policies.

MATHIAS EKSTEDT received the M.Sc., Ph.D., 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. In particular, the research is applied in the power industry and their SCADA and Industrial Control Systems. He is also the Co-Founder of foreseeti, a start-up company developing products for automated and model-based attack simulations.