Journal Pre-proof

Survey: Automatic Generation of Attack Trees and Attack Graphs

Alyzia-Maria Konsta, Alberto Lluch Lafuente, Beatrice Spiga and Nicola Dragoni

PII: S0167-4048(23)00512-6
DOI: https://doi.org/10.1016/j.cose.2023.103602
Reference: COSE 103602

To appear in: *Computers & Security*

Received date: 14 March 2023
Revised date: 25 September 2023
Accepted date: 14 November 2023

Please cite this article as: A.-M. Konsta, A. Lluch Lafuente, B. Spiga et al., Survey: Automatic Generation of Attack Trees and Attack Graphs, *Computers & Security*, 103602, doi: https://doi.org/10.1016/j.cose.2023.103602.

Survey: Automatic Generation of Attack Trees and Attack Graphs

Alyzia-Maria Konsta[a], Alberto Lluch Lafuente[a], Beatrice Spiga[a], Nicola Dragoni[a]

[a]*DTU Compute, Kongens Lyngby, Denmark*

Abstract

Graphical security models constitute a well-known, user-friendly way to represent the security of a system. These classes of models are used by security experts to identify vulnerabilities and assess the security of a system. The manual construction of these models can be tedious, especially for large enterprises. Consequently, the research community is trying to address this issue by proposing methods for the automatic generation of such models. In this work, we present a survey illustrating the current status of the automatic generation of two popular kinds of graphical security models: Attack Trees and Attack Graphs. The goal of this survey is to present the current methodologies used in the field, compare them, and present the challenges and future directions to the research community.

Keywords:
Graphical Security Models, Attack Trees, Attack Graphs, Automatic Generation, Threat Modelling, Survey

1. Introduction

During the last few decades, the use of electronic devices has spread significantly. Companies and individuals are using technology for both personal and professional reasons. In the last decade, with the use of IoT devices in all aspects of our everyday lives, a huge amount of personal and sensitive data has been stored or processed on computer networks. These kinds of devices are limited in resources, and in some cases, the cryptographic algorithms are not applicable. Along with the extended use of electronic devices, the attempts by malicious users to steal this data have increased. As a conse-

quence, the research community has focused on finding ways to protect our data from malicious actors.

One well-known solution for assessing the security of a system is the use of graphical security models. These models represent security scenarios and help security experts identify the system's weaknesses. Their graphical representation constitutes a user-friendly way to analyze the security of the system. In the scope of this paper, we are going to examine only two kinds of these graphical representations: Attack Trees [1] and Attack Graphs [2]. For a comprehensive overview of all the available formalisms, we refer the reader to [3]. Given our focus, we will use the term "Attack Models" to refer to the class of graphical security models constituted by both Attack Trees and Attack Graphs.

Currently, security experts manually produce graphical security models. This procedure, especially for such large systems used nowadays, can be tedious, error-prone, and non-exhaustive. Consequently, the need for automatic procedures arose, and the research community has focused on addressing this issue. Especially in the last decade, researchers have been studying different methods to automatically produce graphical security models. In this work, we present an exhaustive survey of the automatic generation of Attack Models. We provide an overview of the field and identify current challenges and future research opportunities. To our knowledge, this is the first survey paper focusing only on the automatic generation of Attack Models.

Scope. Among all known graphical models of security, we restrict our attention to Attack Trees and Attack Graphs, due to their popularity in the scientific community. The popularity of Attack Trees over alternative models is claimed in various papers, starting with the survey of Kordy et al. [3]. The survey compares Attack Trees to other graphical models of security, namely Attack Graphs, models based on Petri Nets such as "privilege graphs" and models based on UML such as "mal-activity diagrams". The popularity of Attack Graphs and Attack Trees is also evident in bibliographic databases. For instance, the query "attack tree" in Google Scholar yields 7,620 papers, "attack graph" yields 9,210 papers, "privilege graphs" yields 174 papers; and "mal-activity diagrams" yields 186 papers. The prevalence of Attack Trees and Attack Graphs is more evident if the query is combined with "threat modeling".

Research Questions. The main research questions in our work are

- **RQ1:** What kinds of techniques have been proposed for automatically generating Attack Models?
- **RQ2:** What are the characteristics of the proposed approaches in terms of formal rigor, empirical evidence, reproducibility, scalability, etc.?
- **RQ3:** What are the challenges and opportunities in this area of research?

We believe that addressing these questions will help identify current limitations on existing techniques and gaps in the literature in order to provide future directions to the research community. Moreover, addressing these questions will help readers navigate through the literature on automated generation of attack models in a way that makes it easy to see how techniques relate to each other, which tools are available, and ultimately how to identify which method suits their case best.

Contributions. The main contributions are

1. A structured literature review of existing approaches to automated generation of attack models, which addresses **RQ1** and **RQ2** by discussing all existing papers under a common categorization framework. As we shall see, one of the main outcomes of our paper is that the existing approaches generate attack models from (combinations of) *models* of the system under study, *analysis results* of the system under study, or *well-known vulnerability patterns*. We examine all papers through these lenses and look into key aspects of those as demanded by **RQ2**.
2. An identification of limitations and future directions for the research community in order to reduce the gap between research developments and practitioners' needs, therefore answering **RQ3**. One of our main outcomes is a general lack of reproducibility aspects and open tool availability in the proposed approaches.

The paper is structured as follows: Section 2 presents the related work, Section 3 presents the main concepts of Attack Models, Section 4 presents the categorization framework used to classify the papers under study, Section 5 presents the research method used to structure our research, Section 6 presents the classification of the papers, Section 7 discusses limitations, challenges and future research directions, and Section 8 concludes the paper.

Table 1: Overview of the papers included in the Related Work

Paper	After 2015	Automatic Generation	Techniques' Classification	Challenges
[3]			✓	✓
[4]	✓			✓
[5]	✓	✓		✓
Our Work	✓	✓	✓	✓

2. Related Work

In this section, we discuss other surveys that consider the issue of the automatic generation of Attack Models. We chose these works since they provide surveys on Attack Models, and some of them also consider their automatic generation. We also include a table to summarize the main characteristics of the papers included in the Related Work section. In Table 1 four different characteristics are presented. We denote the symbol ✓when the referred paper fulfills the corresponding characteristic. The first column includes a reference to the paper under examination; the second column, "After 2015" refers to the papers that have been published after 2015. This is an important characteristic since more than half of the papers included in our survey were published after 2015. The third column, "Automatic Generation" refers to the papers that study the automatic generation of Attack Models, the fourth column refers to whether the paper presents a classification for the papers included in their survey based on the techniques used in each paper; and the last column indicates whether the paper indicates the challenges identified in the field. The survey papers that we have identified are the following:

Kordy et al. [3] presents a very comprehensive survey of DAG-based graphical models for security. Their work summarizes the state of the art of the existing methodologies, they compare them and classify them based on their formalism. Although this is a very extensive survey, they do not focus on the automated generation of these models. Since it does not cover works published after 2015, their work does not cover several papers on Attack Model generation, that are instead covered in our work.

Lallie et al. [4] examine the effectiveness of the visual representation of Attack Models. They analyze how these structures represent the cyberat-

tacks. They conclude that there is no standard method to represent the Attack Models and that the research community should turn their attention to standardizing the representation. Although this paper is a great contribution, it does not examine the issue of the automatic generation of these structures.

Wojciech et al. [5] provides a survey of the application of formal methods on Attack Trees, their semi-automated or automated generation, and their quantitative analysis. Although they consider automatic generation, their research is not only focused on that. Also, they do not consider the automatic generation of Attack Graphs. Moreover, this paper considered only formal methods approaches and hence does not cover some of the papers we have studied, for example, machine learning approaches.

In our work, we discuss the different methodologies applied by the research community for automating the generation of Attack Models. To our knowledge, this is the first survey focusing on the automatic generation of Attack Models. To be more specific, the automatic generation of Attack Trees is only studied in [5], where all the papers referring to Attack Graphs are not considered- [6, 7, 8, 9, 10, 11, 12, 2, 13, 14, 15], we consider all of the above papers. Also, the following papers that generate Attack Tree are not considered- [16, 17, 18, 19, 20] in [5], on the other hand, we consider all the papers mentioned above. Overall, our survey covers 15 papers that are not covered in any of the existing surveys.

3. Background

Before starting our survey we recall here the basic concepts of Attack Models.

3.1. Attack Trees

An Attack Tree is a graphical representation of potential attacks on the system's security in the form of a tree. Initially introduced by Schneiner [1], this type of representation enables developers to identify the vulnerabilities of the system and facilitates the implementation of countermeasures. They model attack scenarios presented in a hierarchical way, with each labeled node corresponding to a sub-goal of the attacker and the **root** being the main one—the global goal of the attack. The rest of the labeled nodes can be either **children of a node**, a refinement of the parent's goal into subsidiary goals, or **leaf nodes**, representing attacks that cannot be further refined,

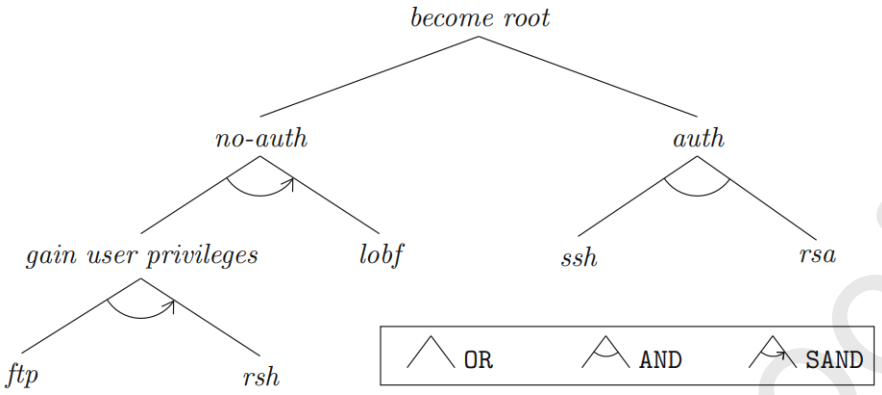

Figure 1: Attack Tree from [21]

also called basic actions. An Attack Tree is a 3-tuple $(N, \rightarrow, n_0)$, where N is a finite set of nodes, $\rightarrow$ is a finite acyclic relation of type $\rightarrow \subseteq N \times M(N)$, where $M(N)$ is the multi-set of N, and n_0 is the root node, such that every node in N is reachable from n_0 [22].

The basic formal model of Attack Trees incorporates two types of refinements: `OR` and `AND`. `OR` nodes represent disjunction (choice), where the parent node's goal is achieved when at least one of the children's sub-goals is achieved, `AND` nodes represent conjunction (aggregation), thus requiring all children's sub-goals to be fulfilled. Several variants of Attack Trees have been proposed. For example, the sequential conjunction refinement `SAND`, is similar to `AND` but requires a specific sequential realization of the children [5, 22]. One example of an Attack Tree is illustrated in Figure 1. In summary, there are two separate ways for the attacker to accomplish their goal, namely, by becoming a root: with or without authentication. The two refined authentication options are ssh and rsa, and both must be fulfilled since they are under an `AND` operator. On the other hand, without authentication, the user must first gain user privileges, fulfill actions ftp and then rsh. Following the acquisition of privileges, lobf should be achieved. The tree notation is very appealing and convenient for a threat analysis process since it can include multiple attacks derived from physical, technical, and even human vulnerabilities [5].

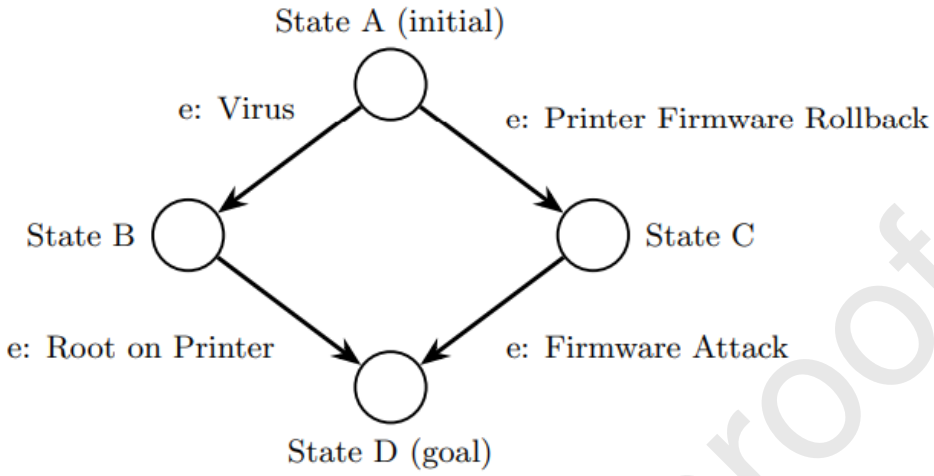

Figure 2: Attack Graph from [23]

3.2. Attack Graphs

Attack graphs are graphical representations that depict all the possible paths an attacker can follow to achieve their malicious goal. They are frequently used to represent complex attacks that have multiple paths and goals [4]. Attack Graphs directly show the existence of vulnerabilities in the system and how attackers can exploit these vulnerabilities to implement an effective attack [24].

An Attack Graph or AG is a tuple $G = (S, \rightarrow, S_0, S_s)$, where S is a set of states, $\rightarrow \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, and $S_s \subseteq S$ is a set of success states [7]. Intuitively, S_s denotes the set of states where the intruder has achieved their goals. Unless stated otherwise, we assume that the transition relation τ is total. We define an execution fragment as a finite sequence of states $s_0, s_1, ..., s_n$ such that $(s_i, s_i + 1) \in \tau$ for all $0 \leq i \leq n$. An execution fragment with $s_0 \in S_0$ is an execution, and an execution whose final state is in S_s is an attack, i.e., the execution corresponds to a sequence of atomic attacks leading to the intruder's goal [7].

One example of an Attack Graph can be seen in Figure 2. The example represents the Attack Graph of a simple network. There is one switch connected to the internet on one side and to a workstation and a printer on the other end. The Attack Graph represents different states of the network, and the edges are exploits that an attacker can use [23]. In the specific example,

the virus can exploit a particular vulnerability in the network in order to transition to State B, where it can exploit another vulnerability to gain root access.

3.3. Attack Trees vs Attack Graphs

A graph can be seen as a set of objects connected by a set of edges. A graph can be either directed or undirected. A directed graph has edges that represent a specific direction; one node is set to be the origin and the other to be the destination. The edges in an undirected graph do not specify which node is the origin and which is the end point; they just denote a two-way connection.

A tree is a special case of a graph, more specifically a Directed-Acyclic Graph (DAG). A tree is represented by three different types of nodes: the root node, which does not have any parents; the internal nodes, which have both a parent and at least one child; and the leaf nodes, which do not have children. So, the tree represents a hierarchy between the nodes.

Taking the above into consideration, we can conclude that the basic structure of Attack Trees and Attack Graphs is different. The Attack Graph can represent more than one goal node, or in some cases, even circles, if an attacker is trying the same unsuccessful action multiple times. On the other hand, the Attack Trees only represent one goal node and constitute an acyclic graph by definition. We can also observe the different visual representations in Figure 1 and Figure 2. Furthermore, in Attack Graphs, the event flow is represented top-down, while in the majority of Attack Trees, this is depicted in a bottom-up way [4]. Moreover, in the same sense, Attack Trees generally use vertices to represent exploits and not preconditions, while preconditions are assumed to have been met in the transition from one exploit to the next. Attack graphs represent both [4]. Essentially, Attack Models have a graph-based structure. The main differences are: how the event flow is depicted, the representation of full and partial attacks, and the representation of preconditions.

4. Categorisation Framwework

In order to provide a structured view of the papers through a common lens, we use the categorization framework visualized in Figure 3. The framework has resulted from a careful analysis of all papers with the aim of identifying commonalities among the proposed approaches. We have indeed observed

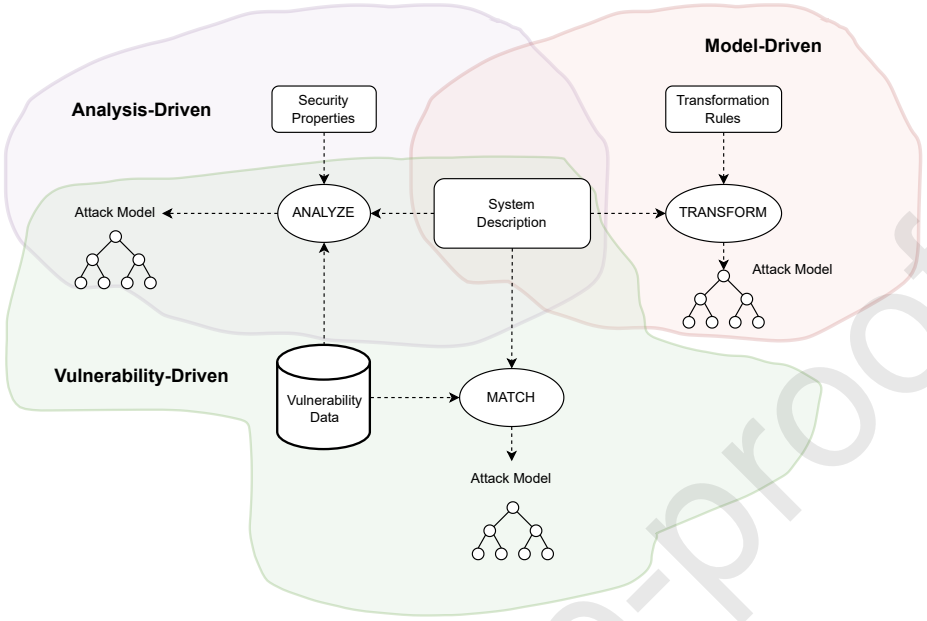

Figure 3: Overview of the categories

that most papers obtain attack models based on system descriptions or models (Model-Driven approaches), existing vulnerabilities (Vulnerability-Driven approaches), and system analysis results (Analysis-Driven approaches). We will describe these "categories" in detail in Section 4.1.

4.1. Categories

Figure 3 provides an overview of the main categories of approaches and their ingredients: Round boxes represent information, while ovals represent processes. The dashed arrows indicate how the processes combine information to produce the attack models. The techniques vary, of course, depending on the specific characteristics of the visualized components. For example, the description of the system can be given in multiple formats, depending on the approach.

As we can see in Figure 3 all the techniques need some information about the system, but they incorporate additional information (analysis results, known vulnerabilities, etc.) to derive the Attack Model. The three categories (Model-Driven, Analysis-Driven, and Vulnerability-Driven) are not disjoint, and some papers may fall into more than one category, as illustrated in Figure 3.

We now describe each category separately.

- *Model-Driven:* This class of approaches uses the description of the system as a starting point. Such a description is directly translated into an attack model, typically using some transformation rules.
- *Analysis-Driven:* This class of approaches also uses the description of the system as a starting point. Such a description is given as input to a process that analyzes it, for example, checking some specific security property or detecting the occurrence of known vulnerabilities. The results of the analysis are used to synthesize the Attack Model.
- *Vulnerability-Driven:* This class of approaches starts with collecting relevant data from a vulnerability database or exploiting vulnerability templates or patterns of attack models. The patterns are then matched to the system description in a process that is often iterative.

5. Research Method

In this section, we present the procedure we followed in order to identify the relevant papers to be included in the survey. More precisely, we decided to conduct our research using the snowballing technique [25]. The snowballing technique refers to the procedure of identifying relevant papers from the reference list or the citations of a selected paper. Our initial set of papers was obtained as described in Section 5.1. The snowballing iterations are described in Section 5.2. Our final pool resulted in 20 papers.

5.1. Start Set

The first step of the snowballing technique is forming the starting set of papers. For this purpose, we had to identify relevant keywords to form a query for a selected database. The keywords we selected to use are: "Automatic Generation", "Automated Generation", "Attack Trees" and "Attack Graphs". We performed our research in DTU Findit `https://findit.dtu.dk`, which is an open (guest access) database and includes publications from well-known journals and databases. We used the following query: title:("Automatic generation" OR "Automated generation") AND title:("Attack trees" OR "Attack graphs"). Our search returned 13 papers, of which 3 were not available online. After reading the papers in detail, we discarded five of them for not being relevant. Therefore, we formed our start set, including five papers. [26, 24, 27, 7, 28].

5.2. Iterations

After finding the start set, we have to decide which papers we are going to include in our final pool. For this purpose, we applied *Backward and Forward Snowballing*. Backward snowballing refers to the examination of the reference list of the papers. In order to identify if a paper will be included, we extract some information regarding the title, the author, and the publication venue. Naturally, we should also take into account the context in which the paper is referenced. At this point, if a paper is still in consideration, we read the abstract and other parts of the paper in order to decide if the paper will be included. The forward snowballing is conducted in order to identify papers from the citation list of the paper being examined. Again, for each paper in the citation list, we extracted some basic information, we took into consideration in which context the citation is taking place, and for the final decision, we read parts of the paper [25].

6. Classification based on the categories

In this section, we present our findings for each category. At the beginning of each category, we present a brief overview of our findings, followed by a detailed presentation of each paper. We also include Tables 2- 4 to graphically represent the key characteristics of every paper. The first column includes a reference to the paper under examination, and the rest of the columns follow the flow of Figure 3. First, the system description refers to the format of the input system information. For the Model-Driven approaches, the third column summarizes the transformation rules. For the Analysis-Driven, we summarize the security properties and the technology used for the analysis. For the Vulnerability-Driven, the vulnerability data used and which procedure was followed for the instantiate process. Finally, for all the categories, the final column represents the format of the Graph Model produced.

6.1. Model-Driven Approaches

All papers falling into this category use a system description as a starting point and then use a set of rules to translate the description into an Attack Model. In comparison with Analysis- and Vulnerability-Driven approaches, the methods in this category do not base the translation on any sophisticated analysis of the security properties of the system or on well-known vulnerabilities. The main reason for this is that the models used as a starting point are

Table 2: Overview of the papers included in the category *Model-Driven*. Abbreviated columns are: (A)pproach and M(odel).

A	System Description	Transformation Rules	M
[29, 30]	• A TREsPASS Socio-technical model • A specific asset to be protected • A profile of the attacker	• Feasible ways to gain access to assets	AT
[31]	• An Attack Tree	• Tree to graph rules for actions and refinements	AG

already in the form of a threat model (in some cases, one of the two kinds of attack models that we consider).

Papers [29] and [30]. The works by Gadyatskaya [29] and Ivanova et al. [30] present approaches to Attack-Defense tree generation for TREsPASS socio-technical models [32]. The approaches are essentially the same, with the main differences being that the work of [30] discusses aspects of the mobility of malicious agents in the system, while the work of [29] includes a discussion on countermeasures.

- **System Description.** A socio-technical representation of the system is based on a simplified version of the TREsPASS model [32]. The model is essentially a graph of locations, assets, and actors, enriched with information about their connections and access control policies.
- **Transformation Rules.** The transformation rules are based on the conditions (sets of actions) needed for agents to gain access to assets based on the specified connections and locations. This may include gaining access to additional assets.
- **Transformation and Attack Model Generation.** The rules are applied to the socio-technical model to obtain a set of so-called attack bundles, which represent feasible attacks on individual assets. Bundles are then combined to generate one Attack Tree. The main idea is that gaining control of one asset could require gaining control of additional assets. Hence, the corresponding bundles are attached to the leaves. The authors of [29] also discuss how to enrich the model with countermeasures.

Paper [31]. The authors are proposing a transformation from Attack Trees to Attack Graphs.

Table 3: Overview of the papers included in the category *Analysis-Driven*. Abbreviated columns are: (A)pproach and M(odel).

A	System Description	Security Properties or Vulnerability Data	Analysis	M
[7, 8]	• A NuSMV model of the network and the intruder	• A temporal logic formula • The CVE database	Model Checking	AG
[9, 10]	• Formal security AADL model • A security property	• A temporal logic formula	Model Checking	AG
[14]	• A set of event logs • A PDDL model of vulnerabilities	• CVE database	Planning	AG
[15]	• A network map	• A specific resource in the network • Data from Nessus, Sidewinder, Checkpoint • CVE & NVD DBs	Graph Reachability	AG
[20]	• A network map	• Undesired reachability in the network	Graph Reachability	AT
[28]	• A Quality Calculus protocol	• Protocol control points to avoid	Static Analysis	AT
[33]	• A description of a building • A model of attacker capabilities	• An asset in the building	Model Checking	AT
[34]	• A network Configuration • Configuration of Machines • Attack exploits	• Logical description of attack	Logical Inference	AG

- **System Description.** An Attack Tree.
- **Transformation Rules.** A set of formally defined rules that apply to each case in the tree: leaves representing attack actions and refinements obtained with each attack tree operator (AND, OR), SAND, etc. The rules specify how to transform the corresponding attack tree pattern into the corresponding representation in an attack graph. For example, a leave/action is transformed into an edge with the name of the action on it.
- **Transformation and Attack Model Generation.** The transformation rules are recursively applied to the tree to obtain the corresponding attack graph.

6.2. Analysis-Driven Approaches

The papers falling into this category use a process to analyze the input information in order to derive the Attack Model. The input information always includes a description of the system, and on top of that, there might be some additional information needed, such as a specification of security properties or data from a vulnerability database. The information is then analyzed to produce results that are then turned into an attack model.

The papers differ in the kind of language used to describe the system and the underlying model that abstracts part of the system, the kind of security properties for which one wants to identify possible threats, the analysis techniques used, and the kind of attack model generated. Most papers use a sort of feasibility analysis to extract possible ways of enacting a threat, which

is then summarized in the attack model. The kind of analysis can range from simple reachability in a graph to more sophisticated analysis based on model checking, constraint satisfaction, or planning, where concrete threats are specified using some suitable language, typically based on logic.

Papers [7, 8]. Sheyner et al. [7, 8] present a tool-supported approach that generates Attack Graphs for given computer networks using the model checker nuSVM [35].

- **System Description.** The system is described as an NuSVM model that specifies several components: a set of hosts, a connectivity relationship, a trust relationship among the hosts, an intruder model, a set of individual actions the intruder can exploit to design an attack, and an intrusion detection system.
- **Security Properties.** Security properties are specified as NuSMV temporal properties and data from the Common Vulnerabilities and Exposures (CVE) database.
- **Analysis and Attack Model Generation.** The model checker takes as input the property and the model M (the network). If the property is satisfied, the model checker returns true; otherwise, it provides a counterexample. The counterexample depicts a path that the attacker can follow to violate the security property. The paths generated through the analysis step, are then combined to generate the Attack Graph.

Papers [9, 10]. Ibrahim et al. [9, 10] present a model-checking based approach for generating Attack Graphs from AADL [36] architectural models of computer and control SCADA networks.

- **System Description.** The description of the system's architecture is specified in the AADL Language [36].
- **Security Properties.** Security properties are specified with CTL temporal logic formulas expressing the compromise of an asset in the architecture.
- **Analysis and Attack Model Generation.** The analysis takes as input the model of the architecture and the security property. The mode checker JKind `http://loonwerks.com/tools/jkind.html` is used to check security properties and to automatically produce counterexamples. Counterexamples are then used to generate the Attack Graph.

Paper [14]. Bezawanda et al. [14] present a planning-based tool that automatically generates Attack Graphs from vulnerability databases.

- **System Description.** The system description consists of a set of event logs and a PDDL [37] specification of a planning domain and problem. The approach uses natural language processing to produce the PDDL specification from the textual description of vulnerabilities from the Common Vulnerabilities and Exposures(CVE) `https://cve.mitre.org/` or the National Vulnerability Database (NVD) `https://nvd.nist.gov/`.
- **Security Properties.** The security property is specified as a goal in the PDDL planning problem.
- **Analysis and Attack Model Generation.** The event logs of the system are used to bind the abstract variables of the PPDL planning domain to form a concrete PPDL problem for the specific system. The PDDL problem is then solved with the help of the PDDL planner, which tries to find a plan, i.e., a sequence of actions the attacker must perform to achieve its goal. Planning solutions are then combined to produce an Attack Graph.

Paper [15]. Ingols et al. [15] present an approach to generate attack graphs for computer network vulnerabilities.

- **System Description.** The system description is in the form of a map of the computer network under study.
- **Security Properties.** The security properties under study are the reachability of specific resources in the newtork.
- **Analysis and Attack Model Generation.** The analysis computes a reachability matrix for the network, which indicates how network resources can be reached from each other. The analysis also imports data from Nessus `http://www.nessus.org`, Sidewinder, and Checkpoint firewalls, as well as from the CVE repositories`http://cve.mitre.org` and NVD `http://nvd.nist.gov`.. From such data and from the network, the analysis generates multiple so-called prerequisite graphs, which consist of three different kinds of nodes: state nodes, prerequisite nodes, and vulnerability instance nodes.

Paper [20]. Hong et al. [20] present an approach to summarize all attack paths in a computer network as an Attack Tree.

- **System Description.** The system description is in the form of a specification of the computer network under study.
- **Security Properties.** A specific node in the network to be reached by the attacker.
- **Analysis and Attack Model Generation.** The analysis computes all possible ways for an attacker to reach the target node in the network. Such paths are then combined into an Attack Tree. After the construction of the full Attack Tree, which users can shrink or expand based on a selection of suitable parameters.

Paper [28]. Vigo et al. [28] propose a technique to extract attack trees from protocols used in distributed systems specified in the Quality Calculus [38] language using static analysis.

- **System Description.** The system description is assumed to be given using the Quality Calculus [38] (an extension of the π-calculus). Such a description specifies how several processes interact via message-passing and how they process the information received using pattern matching on data.
- **Security Properties.** The main security property provided as input is the reachability of a specific program point l in the protocol, typically an unfortunate state that one wishes to avoid.
- **Analysis and Attack Model Generation.** The main idea of the analysis is to determine the possible ways to access a specific program point l. The backward chaining exposes all the paths leading to point l, thus unveiling all the conditions that the data being sent must satisfy in order to reach that l. The paths leading to the program point l are then combined in an Attack Tree.

Paper [33]. Pinchinat et al. [33] present ATSyRA, an Eclipse tool available at `https://atsyra2.irisa.fr/` to generate Attack Trees for military buildings.

- **System Description.** A description of a building and of the attacker's capabilities, is specified as a formal model.
- **Security Properties.** An attack objective, i.e. an asset of the building to be compromised by the intruder.
- **Analysis and Attack Model Generation.** The analysis is based on a model checking the property of compromising the asset and generating an Attack Graph as an intermediary step, which specifies the possible paths of actions needed to reach the asset. Based on the Attack Graph, the security expert specifies a set of high-level actions (HLA), which describe how each attacker action in the graph can be refined into sub-actions. The final step of the analysis uses the HLA actions and the attack graph to construct the Attack Tree.

Paper [34]. Ou et al. [34] present an approach based on MulVAL [6], a system for automatically identifying security vulnerabilities in networks. The central notion of MulVAL is that the configurations of a network and its machines are represented as Datalog tuples and security semantics and attacks are represented as Datalog rules.

- **System Description.** The system description is in the form of a Datalog specification of the network configuration, the configuration of machines, the semantics of security aspects of machines, and the description of attack exploits.
- **Security Properties.** The security properties for which one wants to obtain an attack model are specified with logical rules.
- **Analysis and Attack Model Generation.** The proposed method then analyzes the input by evaluating the Datalog rules, using the MulVal reasoning engine, which produces a set of traces realizing the attacks. The traces are sent to a graph-building algorithm, which combines them to produce an Attack Graph.

6.3. Vulnerability-Driven Approaches

Table 4: Overview of the papers included in the category *Vulnerability-Driven*. Abbreviated columns are: (A)pproach and M(odel).

Paper	System Description	Vulnerability Data	Instantiate Match	M
[2]	• A set of templates • A configuration of the system	• A library of Attack-Tree templates • An attacker profile	Pattern Matching	AG
[11]	• A configuration of the network	• A library of Attack-Tree templates • An attacker profile	Pattern Matching	AG
[12]	• A workflow of the system • A security goal	• A library of Attack-Tree templates • An attacker mode	Pattern Matching	AG
[13]	• A network topology • Network information	• A trained model for recognizing attack paths	Deep Learning	AG
[16]	• A computer network map	• A library of Attack-Tree templates	Pattern Matching	AT
[17]	• A generic system description	• An Attack-Tree template	Pattern Matching	AT
[18]	• An event log	• A CFG of valid attack traces	Parsing	AT
[19]	• An initial version of Attack Tree	• A library of annotated attack trees based on NDV and CAPEC	Library	AT
[39]	• A system model • A refinement specification • Intended semantics	• A trace of successful attacks	Successful Attacks	AT

All the papers falling into this category use a process to match some vulnerability patterns to information about the system. In some cases, the system is assumed to be specified in detail (e.g., as the configuration of a computer network), while others assume that the only available information is system logs or human experts that can be queried. The patterns are often Attack Tree or Graph fragments that can be used to refine or extend a current Attack Model. Some approaches are semi-automatic and need human intervention to decide how to instantiate the patterns and how to proceed with the Attack Model refinement, while others are fully automated.

In the following summary of the papers, we have excluded those papers that also fall into the Analysis-Driven category and that use vulnerability data (namely [14, 15, 7, 8]) to avoid redundancies, but they too fall into this category.

Paper [2]. Phillips et al. [2] present an approach to generate Attack Graphs for computer networks.

- **System Description.** The system description is given in the form of the configuration of a computer network.
- **Vulnerability Data.** Vulnerability data is composed of a set of attack graph templates capturing common attacks along with the attacker profiles.
- **Match and Attack Model Generation.** The procedure starts at the target node for the attacker and iteratively tries to match and

instantiate the attack templates. The procedure is repeated until a node in the network is reached, which is considered to be accessible to the attacker.

Paper [11]. Swiler et al [11] propose a method to generate an attack graph for a computer network.

- **System Description.** The configuration of the computer network under study.
- **Vulnerability Data.** Vulnerability data comes in the form of attack graph templates and attacker profiles. The attack templates represent steps of known attacks or strategies for moving from one state to another in a network.
- **Match and Attack Model Generation.** The tool instantiates the attack templates according to the attacker profile and the network configuration. At each instantiation step, when a node in a graph matches the requirements of the template, a new edge is added to the network and new nodes are created. When no more templates can be matched, the process results in an attack graph.

Paper [12]. Tippenhauer et al. [12] present a work to generate attack graphs for workflows.

- **System Description.** The system description is mainly a workflow for the system.
- **Vulnerability Data.** Vulnerability data is based on attack graph templates and attacker models. Templates are generated from the security goal and the workflow.
- **Match and Attack Model Generation.** The approach starts with an attack graph based on a specific security goal for the attacker. The procedure then extends the initial graph every time a template can be matched with the given workflow description.

Paper [13]. Koo et al. [13] introduce a method to support the generation of a Attack Graphs for computer networks using machine learning.

- **System Description.** The system description is given in the form of the network topology and additional system information.
- **Vulnerability Data.** The approach uses machine learning to train a model to recognize attack paths in attack graphs. The training data is based on vulnerability databases.
- **Match and Attack Model Generation.** The process of generating an attack graph consists of using the trained model to recognize attack graph features in the given network topology.

Paper [16]. Bryans et al. [16] introduce a method to generate Attack Trees automatically from the description of a computer network and a set of templates.

- **System Description.** The description of the computer network under study
- **Vulnerability Data.** The vulnerability data is given in the form of a set of attack tree templates that represent possible attacks.
- **Match and Attack Model Generation.** Templates are matched against the network by instantiating the variables in the template with the actual components of the computer network under investigation. The leaves in the resulting matched template can be refined using the same procedure, yielding a complex attack tree. The tool is available at `https://tinyurl.com/uoptgfb`.

Paper [17]. Kumar et al. [17] introduce a library of attack templates that can be used to refine and instantiate for a specific system.

- **System Description.** No specific system description format is required.
- **Vulnerability Data.** The approach proposes a set of layered Attack Tree templates that have been constructed based on the literature on Attack Trees to integrate their common characteristics.

- **Match and Attack Model Generation.** As for several of the approaches discussed in this section, the approach works by refining the templates iteratively based on specific characteristics of the system under study.

Paper [18]. Pinchinat et al. [18] propose a tool (`http://attacktreesynthesis.irisa.fr/`) to build Attack Trees from system logs.

- **System Description.** No actual system description is used, apart from a set of attack traces of the system (logs of events that resulted in a vulnerability).
- **Vulnerability Data.** Vulnerability data is given as context-free grammar that represents valid threat traces.
- **Match and Attack Model Generation.** The context-free grammar is used to parse the trace log and generate an attack tree akin to a parse tree.

Paper [19]. Jhawar et al. [19] present a semi-automatic procedure for creating Attack Trees based on an initially given proposal and alternating manual manipulations with automatic refinements.

- **System Description.** The system description consists of an initial Attack Tree that describes some tentative threats to the actual system.
- **Vulnerability Data.** Vulnerability data consists of a library of annotated attack trees based on the National Vulnerability Database `https://nvd.nist.gov/` and the Common Attack Pattern Enumeration and Classification `https://capec.mitre.org/`.
- **Match and Attack Model Generation.** The automated steps of the process to refine a tree consist of finding instances of the templates in the current tree proposal. Trees are annotated with predicates that determine if an Attack Tree can be attached to another attack tree as a sub-tree. The final Attack Tree is obtained when the manual and automated steps yield a result that is satisfactory for the users.

Paper [39]. The authors in [39] propose a method to generate Attack Trees from successful attacks.

- **System Description.** The system description is actually not provided. Instead, one departs from a set of logs of malicious activities.
- **Vulnerability Data.** Vulnerability data is given by a set of valid refinements of Attack Trees.
- **Transformation and Attack Model Generation.** A greedy heuristic for encoding and decomposing attack traces is used to generate the Attack Tree, according to the specification of valid refinements.

7. Limitations, Challenges and Future Directions

We start describing in Section 7.1 some features called *dimensions* that we used to identify gaps between the existing approaches. We use those dimensions to analyze the surveyed papers in Section 7.2). Based on our analysis we discuss a set of research challenges and future directions in Section 7.3). Our aim is to serve as inspiration for future directions for the research community. Subsequent sections discuss possible connections with other models similar or extending Attack Trees and Attack Graphs, namely fault trees (Section 7.4), attack-defence trees (Section 7.5), and attack-defence diagrams (Section 7.6).

7.1. Dimensions

Orthogonally, our study aims at spotting the gap between the research on this field and its actual adoption in practice and helping the reader identify which technique suits their case best. Hence, having these two points in mind, we selected some features worth studying in each paper in order to address the aforementioned consideration. We call these features "dimensions":

- *Proofs:* Whether a paper includes formal proofs for the algorithms applied for generating the Attack Models. A proper formalization is important in order to have rigorous guarantees about the use of the approach.
- *Open-Source:* Whether a paper includes a reference to the implementation of the proposed solution and if the code is available online in an open-source repository. This dimension is important for prospective

users to understand if the proposed approach is supported by tools that can be used immediately or if they will have to obtain the tools from the authors or, in the worst case, re-implement the tools based on the paper description. This dimension is related to "reproducibility" of the results (discussed below).

- *Reproducibility:* Whether the authors conducted experiments to validate their approach and if the paper provides detail to reproduce those experiments, witnessed by a reproducibility badge. This dimension is important to assess the scientific validity of the experiments and for prospective users to assess how the methods work on their own systems. We observed that none of the papers have a reproducibility badge.

- *Scalability:* Whether the authors discuss the scalability of their solution to show that it can be applied in real-life scenarios.

7.2. Analysis of Dimensions and Limitations

In this subsection, we provide an overview of the limitations we found in the surveyed papers. The summary is provided in Table 5. We denote the symbol ✓ when the referred paper fulfills the corresponding characteristics for Reproducibility, Open-source, Proof, and Scalability. For Reproducibility, ✗ denotes that experiments are provided, but no reproducibility badge is exhibited. For Open-source, ✗ denotes that a tool is available, but ✓ indicates that it is also an open-source project.

The main limitations we have identified are:

- All approaches from the model-driven category depart from an initial threat model. Hence, users must have had a preliminary threat modeling analysis to exploit the approach to derive an Attack Model, which in part defeats one of the main goals of automated Attack Model generation approaches, namely to provide an initial threat model.

- Analysis-driven approaches often suffer from the complexity of the analysis procedures. For example, model-checking approaches suffer from the state-space explosion, which may render these approaches unsuitable for real-life cases.

Table 5: Overview of dimensions and limitations. Abbreviated columns are: (A)pproach, M(odel), (R)eproducibility, (O)pen-source, (P)roof and (S)calability.

A	R	O	P	S
[2]				
[7, 8]	✗		✓	
[9, 10]	✗		✓	✓
[11]				
[12]			✓	
[13]	✗			
[14]				
[15]	✗			✓
[16]	✗	✓		
[17]				
[18]		✗	✓	✓
[19]		✓	✓	
[20]	✗			✓
[28]			✓	✓
[29, 30]				
[31]			✓	
[34]	✗		✓	✓
[33]		✗		
[39]			✓	

- Vulnerability-driven approaches often use static templates to form the Attack Models. Static templates, cannot be updated automatically, so the user should constantly update the library of templates. Templates aim to serve all kinds of systems, and this can lead to generic models. Also, the limited number of templates can produce incomplete Attack Models. Some other approaches in this category use traces of past attacks to extract vulnerabilities. This is specified in the corresponding system, but the logs are not always accessible. Last, templates are often generic and may not apply to specific novel systems.

A generic limitation of all approaches is that automatically obtained models may not be meaningful for the users, which ultimately requires users to play an active role in the refinement of the models.

7.3. Research Challenges and Future Directions

- *Attack Tree Extensions:* In the automatic generation of Attack Trees most papers focus on basic attack trees with *AND* and *OR* operators. Only a few papers are including the *SAND* operator (see Section 3. The *SAND* operator can represent multiple situations that occur in different kinds of attacks [21]. So we believe that it is important to also include this operator in the automatic generation. Likewise, we believe it would be interesting to explore richer models of attack trees with additional operators proposed in the literature, such as $k-out-of-N$ and defensive measures (see detailed discussion of attack-defense trees (Section 7.5) and attack-defense diagrams (Section 7.6)).

- *Dynamic Solutions:* Systems are being upgraded constantly. New variables or configurations can make an Attack Model useless. The need for more dynamic solutions is crucial. Whenever there are some changes in the system, the current version of Attack Models should be updated, helping security experts identify new potential threats. As far as we know, there is only one paper in the literature proposing a more dynamic solution [2].

- *Lack of Formalization:* The use of semantics and proofs is really important. There are some works that do not use proofs or semantics to support their solution. Neglecting semantics can lead to poor or meaningless results. The use of proofs can help the reader better understand

the concepts used and help the researchers expand the field. Semantics can help the community develop the same language concerning the generation of Attack Models.

- *Forensics:* Another field important to security experts is forensics. Forensics can help security experts identify vulnerabilities that were not previously taken into account and secure their systems better. Using forensics, one can generate Attack Models to depict what goes wrong in a system. This procedure requires the logs or traces of a running system.

- *Meaningful Results:* The automatic generation of Attack Models provides a very good tool for security experts. But it might produce meaningless results that are not usable. There are not many works providing proper means to prevent the generation of meaningless results. Mostly semi-automated procedures have been proposed to avoid the generation of meaningless results [19, 18, 17]. Another work that considers it is [39].

- *Prerequisites:* All of the papers require some prerequisites in order to generate the Attack Models. In some cases, this might be tedious for the security experts if the prerequisites require their involvement (semi-automated), but a fully automated procedure might produce meaningless results. Some prerequisites can be the configuration of the system, which is a very important file that an attacker can exploit to take control of the system. So the prerequisites can themselves be a vulnerability. It is important to find the right balance between the number of prerequisites and producing meaningful results. It is also important to examine the nature of the prerequisites and how crucial they can be for the security of the system.

- *Scalability:* Scalability is one of the main characteristics of a tool. Nowadays, we use very large systems to cover our needs. The automatic generation of Attack Trees should be adapted to perform under these circumstances.

- *Attack Defence Trees:* A few papers are investigating the automatic generation of Attack Defence Trees [29]. After the automatic generation of Attack Models, is it natural to investigate the automatic generation of the corresponding defences/countermeasures.

- *Attack Graphs ↔ Attack Trees*: It would be interesting to explore the transformation of Attack Graphs to Attack Trees and vice versa. Pinchinat et al. are exploiting an Attack Graph to produce an Attack Tree [33]. Also, in [31] the authors propose a transformation of an Attack Tree to an Attack Graph. This method can be explored further and tested to produce more concrete results regarding this transformation. Further experiments can be performed, especially on the results of automatic generation techniques.

7.4. Fault Trees

Fault Trees are graphical models that represent how a component failure can lead to a system failure. They were first introduced in the 1960s at Bell Labs. They are Directed-Acyclic Graphs including leaves and gates. The leaves represent component failures and the gates through which these failures can propagate through the system. The most common gates used in Fault Trees are:

- **AND gate:** The output event occurs only if all input events occur.
- **OR gate:** The output event occurs if at least one of the input events occurs.
- **k/N gate:** The output event occurs if at least k out of N input events occur.
- **Inhibit gate:** The output event occurs if the input event occurs, and the condition drawn to the right of the gate also occurs. It is like an AND gate with two inputs [40].

One can see the similarities between Fault Trees and Attack Trees. They both represent the basic event of interest at the top (the root of the tree), which is then refined into actions until basic actions can no longer be refined. These models have many similarities, but some extensions in each model lead them to grow in different directions [41]. Despite the fact that Fault Trees are out of the scope of this paper, we believe that the similarities between these two models can be exploited in order to explore the different techniques used for the automatic generation of Fault Trees [1] and their adoption in Attack Trees.

[1]Several authors cite this survey [42], which we have not been able to obtain from the authors or the publisher

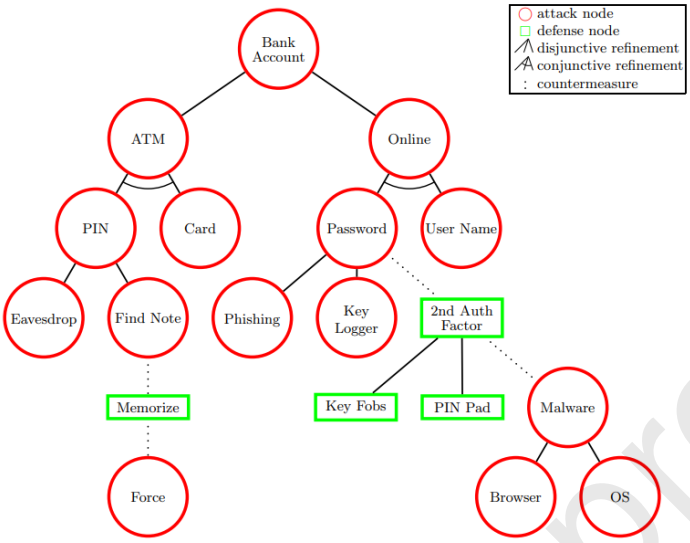

Figure 4: Attack-Defense Tree form [43]

7.5. Attack-Defense Trees

Attack-Defense Trees are an extension of the Attack Trees. They consist of a root node, a set of actions an attacker might take to attack a system, and a set of defense mechanisms the defender can employ. On the Attack-Defense Trees, we have two kinds of nodes, the attack nodes, and the defend nodes, that both can be refined into sub-goals. Also, every node can have a child of the opposite type, called a countermeasure. One example of an Attack-Tree can be seen in Figure 4 [43, 44].

We can observe in Figure 4 that the Attack-Defense Tree uses the same operators as the Attack-Tree. The example depicts a bank account attack. The attack can be achieved either from an ATM or online. We can see in the green rectangles the proposed defenses the defender can employ.

In the scope of this paper, we are not going to examine specifically the automatic generation of Attack-Defense Trees. It is an upcoming field, and it is included in the future directions proposed at the end of the paper. We strongly believe that the automatic generation of Attack-Defense Trees can be the next step in the automatic generation of Attack-Trees/Graphs. We are listing some works focused on the automatic generation of Attack-Defense Trees [29, 45, 46, 47, 48, 49].

7.6. Attack-Defence Diagrams

Attack-Defence Diagrams are acyclic graphs that represent how basic actions (events) can be combined to form a game between the attacker and the defender. The game can be either undecided, a successful attack, or a successful defense. This structure is considered to be more expressive than Attack-Defence Trees. The outcome of the game depends, among others, on parameters like time, cost, and probabilities. These parameters decorate the basic event on the Attack-Defence Diagram. This structure is equipped with more expressive gates than Attack-Defence Trees, including the possibility of repetitive behavior [50]. For further information about Attack-Defense Diagram, we encourage the reader to refer to [50]. Despite the fact that the Attack-Defense Diagrams are out of the scope of this paper, to our knowledge, there is no available research on how to automatically generate Attack-Defense Diagrams. We believe that it would be an asset to the scientific community to further investigate this model, as it offers more expressive ways to represent an attack-defense scenario.

8. Conclusion

We have provided a survey of the state-of-the-art techniques for the automatic generation of Attack Trees and Attack graphs, driven by three main research questions:

- RQ1 What kinds of techniques have been proposed for automatically generating Attack Models? We answered this question by providing a comprehensive and structured literature review of the field. Our survey provides a classification based on the procedure followed to derive the Attack Model. We identified three main categories: Model-Driven approaches, where an input model of the system under study is translated to an Attack Model; Analysis-Driven approaches, where a process analyzes and combines the description of the system with additional information; and Vulnerability-Driven approaches, where existing vulnerabilities are adjusted to the system under study. Based on these categories, our survey describes the specifics of each proposed approach under the same lens, which helps in relating them. .

- RQ2 - What are the characteristics of the proposed approaches in terms of formal rigor, empirical evidence, reproducibility, scalability, etc.? We

answered this question by identifying key dimensions in Section 4 and presenting an analysis of the papers based on the dimensions in Table 5. We consider these dimensions to be important to identify limitations and also to help the reader choose a suitable method to apply.

- RQ3 - What are the challenges and opportunities in this area of research? We answer this question in Section 7 where we discuss the limitations and future directions for the research community. We believe that there is a gap between research and the real employment of the methods, based on our findings about some components, such as scalability and meaningful results. We provide some future directions to the research community to eliminate such a gap.

Taking everything into consideration, we believe that our work has a twofold result: (1) it helps the reader navigate through the literature and decide on the methods and technologies that are more suitable for them, and (2) it provides limitations and future directions for the scientific community.

Acknowledgment

This work has been supported by Innovation Fund Denmark and the Digital Research Centre Denmark, through the bridge project "SIOT – Secure Internet of Things – Risk analysis in design and operation".

References

[1] B. Schneier, Attack trees, Dr. Dobbs journal 24 (12) (1999) 21–29.

[2] C. Phillips, L. P. Swiler, A graph-based system for network-vulnerability analysis, in: Proceedings of the 1998 Workshop on New Security Paradigms, NSPW '98, Association for Computing Machinery, New York, NY, USA, 1998, p. 71–79. doi:10.1145/310889.310919.
URL https://doi-org.proxy.findit.cvt.dk/10.1145/310889.310919

[3] B. Kordy, L. Piètre-Cambacédès, P. Schweitzer, Dag-based attack and defense modeling: Don't miss the forest for the attack trees, Computer Science Review 13-14 (2014) 1–38. doi:https://doi.org/10.1016/j.cosrev.2014.07.001.

URL https://www.sciencedirect.com/science/article/pii/S1574013714000100

[4] H. S. Lallie, K. Debattista, J. Bal, A review of attack graph and attack tree visual syntax in cyber security, Computer Science Review 35 (2020) 100219. doi:https://doi.org/10.1016/j.cosrev.2019.100219.
URL https://www.sciencedirect.com/science/article/pii/S1574013719300772

[5] W. Wideł, M. Audinot, B. Fila, S. Pinchinat, Beyond 2014: Formal methods for attack tree–based security modeling, ACM Comput. Surv. 52 (4) (aug 2019). doi:10.1145/3331524.
URL https://doi-org.proxy.findit.cvt.dk/10.1145/3331524

[6] X. Ou, S. Govindavajhala, A. W. Appel, et al., Mulval: A logic-based network security analyzer., in: USENIX security symposium, Vol. 8, Baltimore, MD, 2005, pp. 113–128.

[7] O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, Automated generation and analysis of attack graphs, in: Proceedings 2002 IEEE Symposium on Security and Privacy, 2002, pp. 273–284. doi:10.1109/SECPRI.2002.1004377.

[8] O. Sheyner, J. Wing, Tools for generating and analyzing attack graphs, in: International symposium on formal methods for components and objects, Springer, 2003, pp. 344–371.

[9] A. T. Al Ghazo, M. Ibrahim, H. Ren, R. Kumar, A2g2v: Automatic attack graph generation and visualization and its applications to computer and scada networks, IEEE Transactions on Systems, Man, and Cybernetics: Systems 50 (10) (2020) 3488–3498. doi:10.1109/TSMC.2019.2915940.

[10] M. Ibrahim, A. Alsheikh, Automatic hybrid attack graph (ahag) generation for complex engineering systems, Processes 7 (11) (2019) 787.

[11] L. Swiler, C. Phillips, D. Ellis, S. Chakerian, Computer-attack graph generation tool, in: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01, Vol. 2, 2001, pp. 307–321 vol.2. doi:10.1109/DISCEX.2001.932182.

[12] N. O. Tippenhauer, W. G. Temple, A. H. Vu, B. Chen, D. M. Nicol, Z. Kalbarczyk, W. H. Sanders, Automatic generation of security argument graphs, in: 2014 IEEE 20th Pacific Rim International Symposium on Dependable Computing, 2014, pp. 33–42. doi:10.1109/PRDC.2014.13.

[13] K. Koo, D. Moon, J.-H. Huh, S.-H. Jung, H. Lee, Attack graph generation with machine learning for network security, Electronics 11 (9) (2022) 1332.

[14] B. Bezawada, I. Ray, K. Tiwary, Agbuilder: an ai tool for automated attack graph building, analysis, and refinement, in: IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2019, pp. 23–42.

[15] K. Ingols, R. Lippmann, K. Piwowarski, Practical attack graph generation for network defense, in: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), 2006, pp. 121–130. doi:10.1109/ACSAC.2006.39.

[16] J. Bryans, L. S. Liew, H. N. Nguyen, G. Sabaliauskaite, S. Shaikh, F. Zhou, A template-based method for the generation of attack trees, in: M. Laurent, T. Giannetsos (Eds.), Information Security Theory and Practice, Springer International Publishing, Cham, 2020, pp. 155–165.

[17] R. Kumar, An attack tree template based on feature diagram hierarchy, in: 2020 IEEE 6th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (DependSys), 2020, pp. 92–97. doi:10.1109/DependSys51298.2020.00022.

[18] S. Pinchinat, F. Schwarzentruber, S. Lê Cong, Library-based attack tree synthesis, in: H. Eades III, O. Gadyatskaya (Eds.), Graphical Models for Security, Springer International Publishing, Cham, 2020, pp. 24–44.

[19] R. Jhawar, K. Lounis, S. Mauw, Y. Ramírez-Cruz, Semi-automatically augmenting attack trees using an annotated attack tree library, in: International Workshop on Security and Trust Management, Springer, 2018, pp. 85–101.

[20] J. B. Hong, D. S. Kim, T. Takaoka, Scalable attack representation model using logic reduction techniques, in: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013, pp. 404–411. doi:10.1109/TrustCom.2013.51.

[21] R. Jhawar, B. Kordy, S. Mauw, S. Radomirović, R. Trujillo-Rasua, Attack trees with sequential conjunction, in: H. Federrath, D. Gollmann (Eds.), ICT Systems Security and Privacy Protection, Springer International Publishing, Cham, 2015, pp. 339–353.

[22] S. Mauw, M. Oostdijk, Foundations of attack trees, in: D. H. Won, S. Kim (Eds.), Information Security and Cryptology - ICISC 2005, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 186–198.

[23] K. Cook, T. Shaw, P. Hawrylak, J. Hale, Scalable attack graph generation, in: Proceedings of the 11th Annual Cyber and Information Security Research Conference, CISRC '16, Association for Computing Machinery, New York, NY, USA, 2016. doi:10.1145/2897795.2897821.
URL https://doi-org.proxy.findit.cvt.dk/10.1145/2897795.2897821

[24] S. Zhong, D. Yan, C. Liu, Automatic generation of host-based network attack graph, in: 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 1, 2009, pp. 93–98. doi:10.1109/CSIE.2009.102.

[25] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, Association for Computing Machinery, New York, NY, USA, 2014. doi:10.1145/2601248.2601268.
URL https://doi-org.proxy.findit.cvt.dk/10.1145/2601248.2601268

[26] K. Muthumanickam, E. Ilavarasan, Automatic generation of p2p botnet network attack graph, in: V. V. Das (Ed.), Proceedings of the Third International Conference on Trends in Information, Telecommunication and Computing, Springer New York, New York, NY, 2013, pp. 367–373.

[27] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, E. i. Tatli, Automated generation of attack graphs using nvd, in: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, CODASPY '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 135–142. doi:10.1145/3176258.3176339. URL https://doi-org.proxy.findit.cvt.dk/10.1145/3176258.3176339

[28] R. Vigo, F. Nielson, H. R. Nielson, Automated generation of attack trees, in: 2014 IEEE 27th Computer Security Foundations Symposium, 2014, pp. 337–350. doi:10.1109/CSF.2014.31.

[29] O. Gadyatskaya, How to generate security cameras: Towards defence generation for socio-technical systems, in: S. Mauw, B. Kordy, S. Jajodia (Eds.), Graphical Models for Security, Springer International Publishing, Cham, 2016, pp. 50–65.

[30] M. G. Ivanova, C. W. Probst, R. R. Hansen, F. Kammüller, Transforming graphical system models to graphical attack models, in: S. Mauw, B. Kordy, S. Jajodia (Eds.), Graphical Models for Security, Springer International Publishing, Cham, 2016, pp. 82–96.

[31] N. D. Schiele, O. Gadyatskaya, A novel approach for attack tree to attack graph transformation, in: International Conference on Risks and Security of Internet and Systems, Springer, 2021, pp. 74–90.

[32] B. Kordy, R. Ivanova, M.G.and Hansen, C. Probst, Trespass d1.3.1 initial prototype of socio-technical security model (2013).

[33] S. Pinchinat, M. Acher, D. Vojtisek, Atsyra: An integrated environment for synthesizing attack trees, in: S. Mauw, B. Kordy, S. Jajodia (Eds.), Graphical Models for Security, Springer International Publishing, Cham, 2016, pp. 97–101.

[34] X. Ou, W. F. Boyer, M. A. McQueen, A scalable approach to attack graph generation, in: Proceedings of the 13th ACM conference on Computer and communications security, 2006, pp. 336–345.

[35] Nusmv: a new symbolic model checker, https://nusmv.fbk.eu/.

[36] Sei, architecture analysis and design language (aadl), https://www.sei.cmu.edu/our-work/projects/display.cfm?customel_datapageid_4050=191439,191439, accessed: 2023-02-08.

[37] N. Ghosh, S. K. Ghosh, A planner-based approach to generate and analyze minimal attack graph, Applied Intelligence 36 (2012) 369–390.

[38] H. R. Nielson, F. Nielson, R. Vigo, A calculus for quality, in: C. S. Păsăreanu, G. Salaün (Eds.), Formal Aspects of Component Software, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 188–204.

[39] O. Gadyatskaya, R. Jhawar, S. Mauw, R. Trujillo-Rasua, T. A. Willemse, Refinement-aware generation of attack trees, in: Security and Trust Management: 13th International Workshop, STM 2017, Oslo, Norway, September 14–15, 2017, Proceedings 13, Springer, 2017, pp. 164–179.

[40] E. Ruijters, M. Stoelinga, Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools, Computer Science Review 15-16 (2015) 29–62. doi:https://doi.org/10.1016/j.cosrev.2015.03.001.
URL https://www.sciencedirect.com/science/article/pii/S1574013715000027

[41] C. E. Budde, C. Kolb, M. Stoelinga, Attack trees vs. fault trees: Two sides of the same coin from different currencies, in: A. Abate, A. Marin (Eds.), Quantitative Evaluation of Systems, Springer International Publishing, Cham, 2021, pp. 457–467.

[42] A. Berres, H. Schumann, Automatic generation of fault trees: A survey on methods and approaches (2016).

[43] B. Kordy, S. Mauw, S. Radomirović, P. Schweitzer, Foundations of attack–defense trees, in: P. Degano, S. Etalle, J. Guttman (Eds.), Formal Aspects of Security and Trust, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 80–95.

[44] B. Kordy, S. Mauw, S. Radomirović, P. Schweitzer, Attack–defense trees, Journal of Logic and Computation 24 (1) (2014) 55–87.

[45] M. Aijaz, M. Nazir, M. N. Anwar, Generating attack–defense tree by automatically retrieving domain-specific security attack patterns, in: M. Pandit, M. K. Gaur, P. S. Rana, A. Tiwari (Eds.), Artificial Intelligence and Sustainable Computing, Springer Nature Singapore, Singapore, 2022, pp. 131–145.

[46] S. Salva, L. Regainia, A catalogue associating security patterns and attack steps to design secure applications, J. Comput. Secur. 27 (2019) 49–74.

[47] M. Eckhart, K. Meixner, D. Winkler, A. Ekelhart, Securing the testing process for industrial automation software, Comput. Secur. 85 (C) (2019) 156–180. doi:10.1016/j.cose.2019.04.016.
URL https://doi.org/10.1016/j.cose.2019.04.016

[48] K. Siu, H. Herencia-Zapana, D. Prince, A. Moitra, A model-based framework for analyzing the security of system architectures, in: 2020 Annual Reliability and Maintainability Symposium (RAMS), 2020, pp. 1–6. doi:10.1109/RAMS48030.2020.9153607.

[49] S. Salva, L. Regainia, A security pattern classification based on data integration, in: P. Mori, S. Furnell, O. Camp (Eds.), Information Systems Security and Privacy, Springer International Publishing, Cham, 2018, pp. 105–129.

[50] H. Hermanns, J. Krämer, J. Krčál, M. Stoelinga, The value of attack-defence diagrams, in: F. Piessens, L. Viganò (Eds.), Principles of Security and Trust, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 163–185.

Alyzia Maria Konsta is a Ph.D. student at the Technical University of Denmark. She received her M.Sc in Electrical and Computer Engineering from the National and Technical University of Athens in 2020. Her research interests are in formal methods, risk assessment, software engineering, and security.

Beatrice Spiga is a computer science M.Sc student at the University of Bologna. During the M.Sc program, she also performed research for her master's thesis at the Technical University of Denmark, on the automatic generation of attack trees. Her research interests are in cybersecurity and risk assessment. However, she is also open to a wide variety of research topics in this field.

Alberto Lluch Lafuente is Full Professor at the Technical University of Denmark. He is currently head of the section for Software Systems Engineering. Previously, he was Associate Professor at the Technical University of Denmark, Assistant Professor at IMT School for Advanced Studies Lucca and Postdoctoral Researcher at the University of Pisa. He obtained his PhD degree from the Albert-Ludwigs Universität in Freiburg in 2003. His research interests are in formal methods, software engineering, artificial intelligence and security.

Nicola Dragoni is Professor in Secure Pervasive Computing at DTU Compute, Technical University of Denmark, where he also serves as Deputy Director for Research, Head of Section (Cybersecurity Engineering) and Head of the DTU Center for Digital Security (DIGISEC). Nicola Dragoni received the M.Sc. (cum laude) and Ph.D. degrees in Computer Science from University of Bologna, Italy. His main research interests centre around pervasive computing and security, with latest focus on Internet-of-Things and Fog/Edge computing. He has co-authored 145+ peer-reviewed scientific papers in international journals and conference proceedings. He has edited 3 journal special issues and 1 book. He has been active in several national and international projects.

Declaration of interests

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

☒ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Alyzia Maria Konsta reports financial support was provided by Innovation Fund Denmark.

Alyzia: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - Original Draft, Writing - Review & Editing, Supervision, Project administration
Alberto: Conceptualization, Methodology, Formal analysis, Investigation, Writing - Review & Editing, Supervision, Funding acquisition
Nicola: Conceptualization, Methodology, Supervision, Funding acquisition
Beatrice: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Writing - Original Draft