Hindawi
Security and Communication Networks
Volume 2021, Article ID 1444182, 18 pages
https://doi.org/10.1155/2021/1444182

Research Article

An Automatic Planning-Based Attack Path Discovery Approach from IT to OT Networks

Zibo Wang,[1,2] Yaofang Zhang,[1,2] Zhiyao Liu,[3] Xiaojie Wei,[1,2] Yilu Chen,[1,2] and Bailing Wang[1,2]

[1]*School of Computer Science and Technology, Harbin Institute of Technology, Weihai 264209, China*
[2]*School of Cyber Science and Technology, Harbin Institute of Technology, Harbin 150001, China*
[3]*China Industrial Control Systems Cyber Emergency Response Team, Beijing 100040, China*

Correspondence should be addressed to Bailing Wang; wbl@hit.edu.cn

Received 30 July 2021; Revised 17 September 2021; Accepted 16 October 2021; Published 31 October 2021

Academic Editor: Yulei Wu

With the convergence of IT and OT networks, more opportunities can be found to destroy physical processes by cyberattacks. Discovering attack paths plays a vital role in describing possible sequences of exploitation. Automated planning that is an important branch of artificial intelligence (AI) is introduced into the attack graph modeling. However, while adopting the modeling method for large-scale IT and OT networks, it is difficult to meet urgent demands, such as scattered data management, scalability, and automation. To that end, an automatic planning-based attack path discovery approach is proposed in this paper. At first, information of the attacking knowledge and network topology is formally represented in a standardized planning domain definition language (PDDL), integrated into a graph data model. Subsequently, device reachability graph partitioning algorithm is introduced to obtain subgraphs that are small enough and of limited size, which facilitates the discovery of attack paths through the AI planner as soon as possible. In order to further cope with scalability problems, a multithreading manner is used to execute the attack path enumeration for each subgraph. Finally, an automatic workflow with the assistance of a graph database is provided for constructing the PDDL problem file for each subgraph and traversal query in an interactive way. A case study is presented to demonstrate effectiveness of attack path discovery and efficiency with the increase in number of devices.

1. Introduction

Since information technology (IT) was introduced into all walks of life, the threat from hackers and virus attacks have never been got rid of. However, it does not prevent industrial enterprises from adopting the commercial-off-the-shelf software and hardware and the general network connectivity into operational technology (OT) networks, such as industrial control networks [1]. The IT/OT convergence provides attackers more opportunities to launch targeted attacks whose consequences can be disastrous against the real physical world. The industrial control security incidents in the past decade are the best proof that cyberattacks are gradually infiltrating from the IT networks to the OT networks [2].

Apart from the cyberattacks migrated from IT networks, some inherent issues exist in the OT networks, such as design defects in industrial control network protocols [3] and vulnerabilities of proprietary devices [4]. On account of frequent interactions between IT devices and OT components, there are no clear boundaries between IT and OT partitions in the current industrial environment. In other words, any compromise that occurred on the devices or networks in either IT or OT side has an undesirable impact on the overall safety and security. Therefore, both IT and OT aspects should be taken into consideration simultaneously for cybersecurity analysis in a comprehensive assessment [5].

In general, the security assessment mostly relies on a standalone vulnerability scanning for services or devices in the IT and OT networks. Although more specific vulnerability information can be obtained by scanning, they are difficult to be used to understand means and intents of

sophisticated attacks, let alone to suit for the comprehensive assessment. To that end, the concept of the attack path is presented to describe an alternative sequence of exploitation steps that can lead to a successful attack [6]. Direct, indirect, and subliminal attack paths are concerned by security practitioners and researchers. The process of generating those paths can be treated as that of simulating specific attack behaviors [7].

Focusing on finding all possible attack paths, dependencies of topologies, vulnerabilities, exploitations, and targets are integrated into a graph, called attack graph [8]. Since the attack graph model was first constructed in 1997, quantities of generation methods for it have been widely used in a variety of scenarios to discovery attack paths [6]. Among them, automated planning, a branch of the artificial intelligence (AI), is adopted in the attack graph generation, which transforms finding valid paths into solving problems of a given attack scenario by a planner [9]. It has two obvious characteristics compared with other attack graph generation methods. For one thing, various mature domain-independent planners on the graph plan algorithm can be utilized to accomplish path discovery tasks. For another, a standardized planning domain definition language (PDDL) has favorable data representation with rich-level semantics and the numerical logic deduction to support the planner, which is suitable for modeling changeable attack scenarios by encoding domain knowledge and variable conditions [10, 11].

Nevertheless, when implementing into the IT and OT networks, the planning-based method also suffers from several problems mainly in three aspects: (1) Despite the advantage of the PDDL descriptions in the modeling, it loses some heterogeneous and scattered information with the abstraction of the attacking knowledge and the network topology, which is unconducive to make sense of attack paths. (2) Most attack graph generation methods are limited by the scalability with the increasing scale of the network topology. Unfortunately, there is no exception for the planning-based method whose bottleneck restrictions exist in parsing the complex PDDL problem file of encoding a complete and larger attack scenario and attack path enumeration to call the planner. (3) Enhancing the automation in each stage is a long-standing topic for the attack graph modeling. Faced with frequent changes in the attack scenarios, a challenge comes from how to reconstruct corresponding PDDL descriptions.

To cope with the problems as mentioned above, we proposed an automatic planning-based attack path discovery approach on the basis of the literature [9]. We aim to extend planning-based method for large-scale attack graph generation. The main contributions of this work are summarized as follows:

(1) We present a formal data description method for attacking knowledge and network topology. Modeling by PDDL still possesses the advantage in descriptions. The combination with a graph data model does favor to globally understand attack paths with the integration of scattered information in the form of entities and relations.

(2) We improve the conventional planning-based attack graph generation method for a large-scale network. A device reachability graph partitioning algorithm is introduced to obtain subgraphs that are small enough and of limited size, facilitating the discovery of solution by planner, and reducing the burden of parsing the PDDL files.

(3) We provide an automatic workflow with the assistance of a graph database. In response to the attack scenarios changing, an automatic construction for the PDDL files is implemented for each subgraph. The graph database with the model makes it possible for subsequent visualization and assessment in an interactive way.

The rest of the paper is organized as follows: A related work is summed up in Section 2. An overview of our proposed approach is given in Section 3. Section 4 provides a formal representation method including the PDDL and a graph data model. In Section 5, 4 algorithms are elaborated to complete a series of tasks on the attack path discovery. A case study demonstrated the function and performance of our proposed approach in Section 6. Finally, we conclude the research in Section 7.

2. Related Work

In this section, we simply review methodologies and techniques to generate various types of attack graphs, such as model checking, deductive reasoning, automated planning, parallel computing, and graph data modeling. We mainly focus on the following two perspectives, namely, the scalability and the formal data representation.

In general, model checking is a technique for determining whether a formal model satisfies a given property or not. By finding counterexamples on attack sequences, paths that breach security attributes are represented in a state attack graph [12]; however, it also confronts exponential state-space problems even used in middle-scale networks. Different from the state attack graph, logical attack graphs are built by deductive reasoning to demonstrate attack steps and their prerequisites for each action [13–16]. A well-known and open-source reasoning framework, called MulVAL (Multihost, Multistage Vulnerability Analysis), is utilized to infer attack paths among attack goals and configuration information [15, 16], which is fit for risk assessment in the large-scale enterprise environment.

Those two kinds of attack graphs belong to the complete graph, probably involving attack paths that cannot reach the attack goal, whereas a minimal attack graph is defined as all attack paths terminated to the specific goal [9]. Planning-based methodologies can be adopted to generate the minimal attack graph [17–20]. A set of domain-independent planners can be used to build attack graphs, such as GraphPlan, FF, Metric-FF, LPG-td, and SGPlan [11]. Nevertheless, it is universal acknowledge that the planning-based methodologies has limitations in scalability. Consequently, a graph reduction method is introduced to simplify the task of searching without sacrificing the quality of attack

paths in [17]. Concentrating on penetration information of a testing goal, a compact planning graph algorithm is proposed in [20] to prune redundant attack path branches for the improvement of the scalability.

Compared to the serial approaches mentioned above, another attack graph generation has a dependence on parallel computing [21–23]. Its core idea is using a partition algorithm to split a large-scale complex network topology and processing each subgraph by multiple agents at the same time, which emphasizes the load balancing to enhance the efficiency of the attack graph generation.

Besides the methodologies and techniques, an appropriate formal data representation of the expert knowledge and the network configuration is important for the attack graph generation as well. It is not only meaningful to describe multisource information in each corresponding modeling approach, but also helpful to promote readability for understanding attack paths. There is a research direction for modifying a generated attack graph. For instance, a process-mining algorithm is employed to extract the chronological order and logical relationship among cyberattack behaviors, generating an attack graph on massive security alerts [24]. Then, the complex graph is split based on some branches without changing its original structure, which makes it easier to understand. In [25], an ontology is constructed for the MP (multiple prerequisite) graphs that scales nearly linearly as the network size growing. It enables network administrators to make sense of the semantics of attack paths by knowledge inferences. In order to build a simplified attack graph, the concept of abstracted visualizations is implemented by aggregating part of the original attack steps according to an asset [26].

Since the input and output information of attack graphs are manifested as connected data, it can be naturally stored as graph data in the form of nodes and edges. With the recent popularity of the graph database, the graph data model is paid more attention by researchers to represent data, such as the topology and vulnerabilities [27–30]. In addition to in consideration of the data storage form, gaining the valid information among the large-scale and isolated data is still an impending demand for building attack graphs. Graph traversal queries can provide an efficiency and scalable solution for the difficulty. In [28], a cyberattack-oriented graph data model is given around attack paths to capture relationships among entities in the security domain allowing for the division of graph models into interconnected layers, which is convenient for supporting other collaborative assessment tasks.

Inspired by the researches listed in this section, we attempt to solve the problems in three aspects described in the introduction part. Our proposed approach innovatively combines two methodologies that are graph partitioning and graph data modeling to mainly promote automation and scalability for the IT and OT network environment.

3. Proposed Method

Our method aims to find attack paths from IT to OT networks in a rapid and automatic way. We applied the PDDL for formal representation to model the network topology and attacking knowledge as inputs of an independent AI path planner. To further improve the scalability of attack path enumeration using the planner, a device reachability graph partitioning is introduced before the attack path planning phase. Subsequently, calling the enumeration algorithm in a multithreading manner for each subgraph, all attack paths can be discovered. As a core component of our method, a graph database has advantages in the aspects of the storage and traversal query. It manages entities and maintains relationships in the phases of information gathering, key element extraction, and attack path planning. Moreover, automatically constructing PDDL files is realized by obtaining property fields lying in the entities. Particularly for the large-scale networks, the interactive query plays a vital role in multisource information on attack paths, and the graph database makes it possible for the attack path visualization or assessment in a limited time.

As illustrated in Figure 1, the proposed framework for attack path discovery consists of information gathering, key element extraction, topology analysis, graph data construction, formal representation, attack path planning, and its application. We accomplish attack path discovery tasks of different phases with the generation and exchange of data workflow. The descriptions of the major function modules are provided as follows:

(i) Topology analysis is as follows: it analyzes basic data from information gathering to parse network reachability among devices, calculating the overall network complexity. According to a preset value of subgraph scale, the network topology in a large scale is split into multiple parts for subsequent attack path planning in a multithreading way.

(ii) Graph data construction is as follows: the data from information gathering, key element extraction, and topology analysis is stored in the form of graph data. By means of fast traversal query APIs in the graph database, it is possible to obtain essential information for both formal representations using the PDDL and attack path application. Besides, the dependency between vulnerabilities, stemming from attack paths, can be added into graph data relations.

(iii) Formal representation is as follows: using the results of the graph database queries, two PDDL files on domain and problem can be generated automatically for the AI planner. The domain is a set of actions which focuses on the state transition of attacking, and the problem contains initial states of devices, vulnerabilities, topology, as well as the goal states.

(iv) Attack path planning: the AI planner generates a shortest attack path based on the specific PDDL files. In order to find all attack paths, an enumeration algorithm is developed for each subgraph, which needs to modify the problem file for replanning paths. When a set of attack paths is

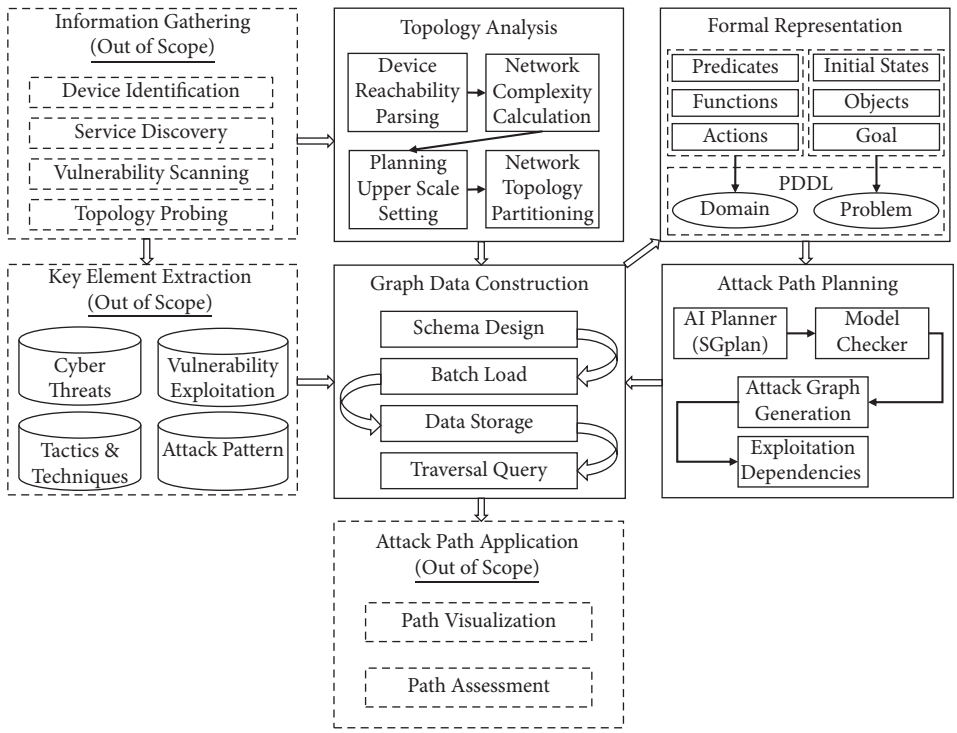

Figure 1: Overview of our automatic planning-based attack path discovery method.

available, a global attack graph is generated for further analyzing exploitation dependencies.

In the following sections, we discuss the main parts of the proposed method in detail.

4. Formal Data Representation

In this section, we formally define a network model, a device reachability, an attack path, and an attack graph. After that, the definitions of domain and problem in PDDL, as well as AI planner, are given. Considering the demands of the attack path generation as aforementioned, a graph data model is constructed to represent entities and relationships in a network.

4.1. Preliminaries. Given a formal network model, the first and core step in building an attack graph is determining its reachability. An attack graph is an abstraction of representing all paths that an attacker is able to exploit interdependencies among existing vulnerabilities:

(i) *Definition 1.* A network model is defined as $N = <D, R, V>$, where D is a set of devices D_i $(i = 1, 2, 3 \ldots)$, $R \subseteq D \times D$ is a set of reachable conditions R_{ij} $(i = 1, 2, 3 \ldots, j = 1, 2, 3 \ldots)$, and V is a set of vulnerabilities V_i $(i = 1, 2, 3 \ldots)$.

(ii) *Definition 2.* A device reachability refers to whether some ports on the device can be accessed via TCP or UDP connections from other devices in the network, which can be analyzed from firewall rules and the network topology. Boolean $R_{ij} = 1$ denotes that D_i reaches D_j via an open port.

(iii) *Definition 3.* An attack path (AP) is a finite acyclic path of a sequence on devices D_i and vulnerabilities V_i, where $\mathbf{AP} = \{(x_1, x_2, \ldots, x_n) | x_n \in (D \times V) \cup (V \times D)\}$, $n = 1, 2, 3 \ldots$.

(iv) *Definition 4.* An attack graph (AG) is a data structure that represents the intersection of all attack paths, where $\mathbf{AG} = \{AP_i | AP_i$ is an attack path in $N\}$, $i = 1, 2, 3 \ldots$.

4.2. Domain and Problem in PDDL. The PDDL and its variants are often used for encoding domain knowledge. Given that the information has been represented into PDDL files of domain action models and problem statements, there are a variety of classical and heuristic planners accessible to generate attack plans:

(i) *Definition 5.* A PDDL domain is an abstract description on a series of problems, including requirements, functions, predicates, and available actions with the preconditions and postconditions. It models a variety of attacks which corresponds to vulnerabilities, tactics, and techniques.

(ii) *Definition 6.* A PDDL problem is a concrete instance of a certain PDDL domain, including objects, initial states with numerical predicates, and a goal state. It models the device reachability, service running on the devices, and attacker abilities, such as privileges, credentials, and even an entry point of a compromised network.

(iii) *Definition 7.* An AI planner is a search algorithm designed for a specific purpose, finding out a plan

that satisfies a PDDL problem. In this paper, we choose a domain-independent planner, called SGPlan (https://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/sgplan5.html), to solve the planning problem in PDDL, which generates a shortest attack path at a time.

Figure 2 depicts our running example that a simple network scenario is modeled by domain and problem using the PDDL and the planner finds out a valid attack path. The simple network scenario contains two devices where they both run services with vulnerabilities. One vulnerability exists in the TIA portal used on engineering workstations (EWS) owing to the improper input validation. The other vulnerability affects the CPU of Siemens S7-300 PLCs where it can be arbitrarily switched to a defect mode. The planning attack path is <Attacker $\longrightarrow$ Improper Input Validation $\longrightarrow$ *EWS* $\longrightarrow$ Improper Control $\longrightarrow$ *PLC*>.

Note that we expect to express that an attacker may send specially crafted packets to gain privileges on the EWS and then exploit the vulnerability of the PLC CPU to take an improper control of it. However, the output of the planner tends to miss some key information, which is used to describe an attack path. For that reason, we next devise entities and relationships in graph data to make up for the missing semantics.

4.3. Graph Data Model. In our approach, we construct a graph data model to represent around devices, networks, and vulnerabilities. It is made up of a set of nodes and edges, where nodes represent entities and edges represent relationships between entities. There are five entities in this model including the Device, the Vulnerability, the Component, the Domain, and the Tactic & Technique. The properties of each entity are shown in Table 1. Facts corresponding to the entities are taken from the phases of the information gathering and the key element extraction. Particularly, the Component is a general term, referring to services, operating systems, or even the hardware, using type field in the property to distinguish. The term "Tactic & Technique" is derived from ATT & CK for ICS (https://collaborate.mitre.org/attackics/index.php/Main_Page) to describe individual techniques under the tactics on some specific vulnerabilities.

In addition, seven relationships are summarized in Table 2. In the initial stage of the construction, the top five relationships described in the table can be provided among the node of facts. However, the relations between the attack and the exploit are added into the graph data model when attack paths are generated by the proposed enumeration method.

As illustrated in Figure 3, we further expand the semantics of the attack path mentioned in the previous section. Take the PLC in the simple network scenario as an example. In the process control domain, the CPU of the PLC may be affected by a vulnerability so that the PLC switches from run mode to defect mode. To that end, attacker gains privilege on the EWS which connects to the PLC and exploits the vulnerability of an improper control, launching the attack to make the PLC denial of service (DoS). As a result, the PLC may inhibit response functions (IRFs) to message feedback by sensors or actuators in the field. To some extent, the combination of the graph data model with the original discovery method is a desirable way to enhance the readability of attack paths.

5. Attack Path Discovery Approach

In this paper, an automatic, multithreading, AI planning-based enumeration approach is proposed as the core attack path discovery algorithm. Based on the device reachability defined in Section 4.1, a graph partitioning method is introduced to generate subgraphs within the preset numbers of nodes. Each subgraph is assigned to a thread and then the planner is called separately to complete the attack path enumeration. Using the graph data model mentioned in Section 4.3, files of the domain and problem in PDDL can be combined automatically with the help of traversal query for properties, which follows the principles of PDDL syntax.

Given a pair of the domain and problem for a specific situation, a fixed shortest attack path can be solved. By modifying the problem in PDDL, all attack paths can be generated. For that end, an attack path enumeration needs to be developed. We improved the enumeration algorithm mentioned in [9], where it can adapt to a larger network scale for the attack path discovery. Once all attack paths are obtained, an attack graph can be built for analyzing the relations of the attack and the exploitation, which need to be added for the device nodes and vulnerability nodes in the aforementioned graph data model. In the following sections, we will give implementations for the graph partitioning (Algorithm 1), the PDDL files combination (Algorithm 2), the attack path enumeration (Figure 4), and the multithreading execution (Algorithm 3).

5.1. Device Reachability Graph Partitioning. Reachability determines the accessibility conditions among the services running on the target devices. With an increase of numbers of devices in a large-scale IT and OT networks, finding attack paths is a huge burden in a limited time. As we know, the original planning and enumerating attack paths are also not suitable for the large-scale networks, due to the time-consuming process of parsing large PDDL files and the repeated path traversal. Naturally, it is an urgent demand for the attack path discovery to divide the device reachability graph into small-scale subgraphs. Subsequently, we use a multithreaded method to find the attack path for each subgraph.

The complex graph segmentation algorithm is presented in [24], which is originally used to enhance the readability of an attack graph. The idea of the algorithm is that it searches for the branch nodes to split the whole graph and complete subgraphs based on their structure. However, we simplify the above algorithm to partition a device reachability graph in this paper. The input contains a device reachability graph and the subgraph size. The output is all subgraphs. More details are shown in the pseudocode of Algorithm 1.

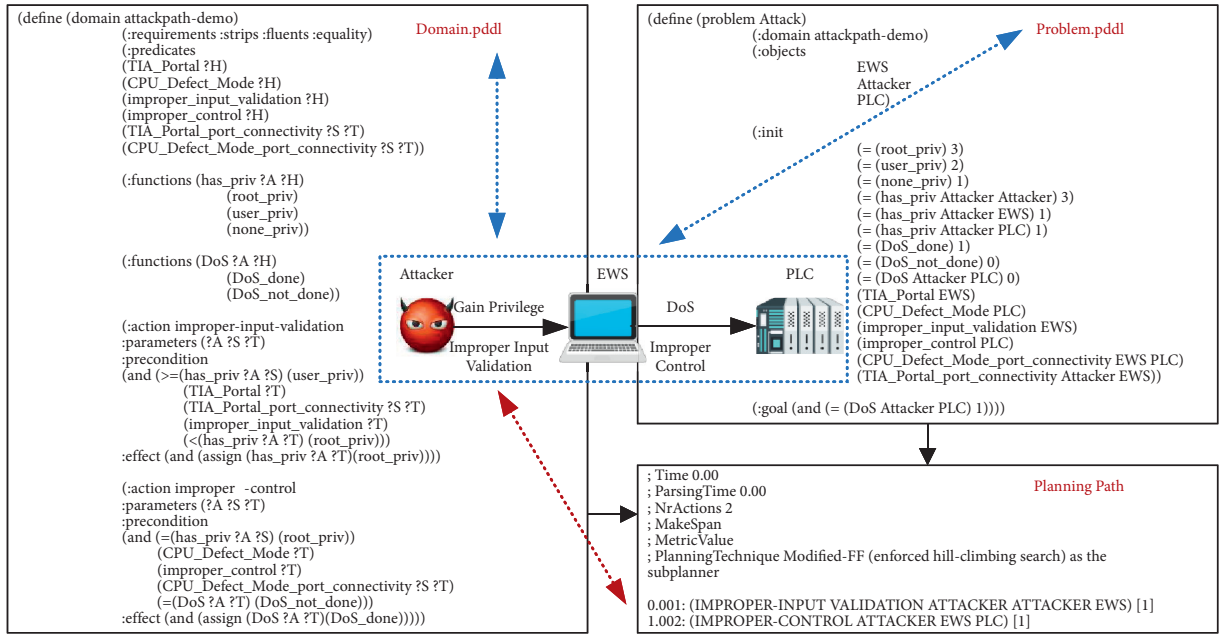

Figure 2: Definition of the domain and problem using the PDDL for a simple network scenario.

Table 1: Entity information in the graph data model.

No.	Entity name	Properties
1	Device	ID, name, type, vendor, version, firmware/OS, fubgraph_ID
2	Vulnerability	ID, name, type, ATT_vector, CVE_ID, precon, postcon, brief_info
3	Component	ID, name, type, vendor, version
4	Domain	ID, name, network, access_rules (service_port)
5	Tactic & technique	ID, name, brief_info, data_source

Table 2: Relation information in the graph data model.

No.	Relation name	Descriptions
1	conn	A reachability exists between two devices.
2	consist	A device belongs to a domain divided by a network.
3	offer	A component is offered by the devices.
4	has	A component has a vulnerability.
5	depend	The tactic and technique depend on a vulnerability.
6	attack	A device is affected by the vulnerability, leading to an attack.
7	exploit	A compromised device exploits a vulnerability in another devices.

5.2. Automatic PDDL Domain and Problem Construction. As discussed in Section 4.2, it is critical for the planner to define domain and problem files. The domain file encodes predicates and actions. The problem file encodes objects, initial states, and goals. When the planner was run for the different subgraphs, respectively, some pairs of domain and problem files are needed at the same time. As a result, it is necessary to construct these two files in an automatic way, particularly for the situation of large amount subgraphs. To realize the above purpose, we build two templates for domain and problem in PDDL. Combining with the query results from the graph database and the templates, domain and problem files can be generated within a short time, even if the reachability and device configuration are modified in the previous phases. More details are shown in the pseudocode of Algorithm 2.

5.3. Attack Path Enumeration. According to a pair of domain and problem PDDL files, the planner provides a solution to find the shortest attack path. Nevertheless, an attack path enumeration strategy is supposed to be developed to look for all attack paths. Referring to the literature [9], a customized algorithm is given by modifying the problem PDDL file to generate new attack paths until an exclusive path set is found. The way of modifying is to automatically

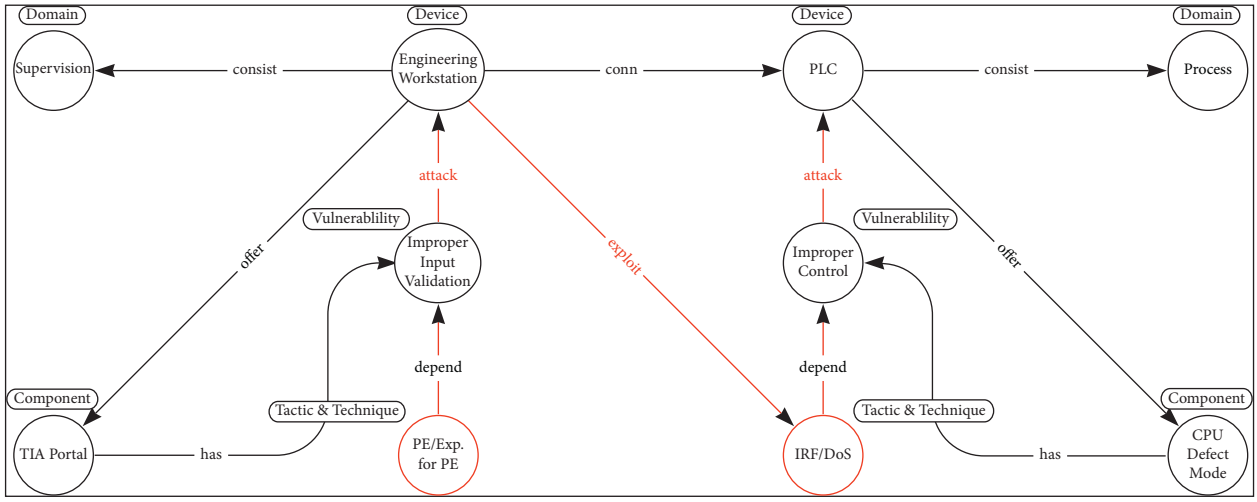

Figure 3: Graph data model for a simple network scenario.

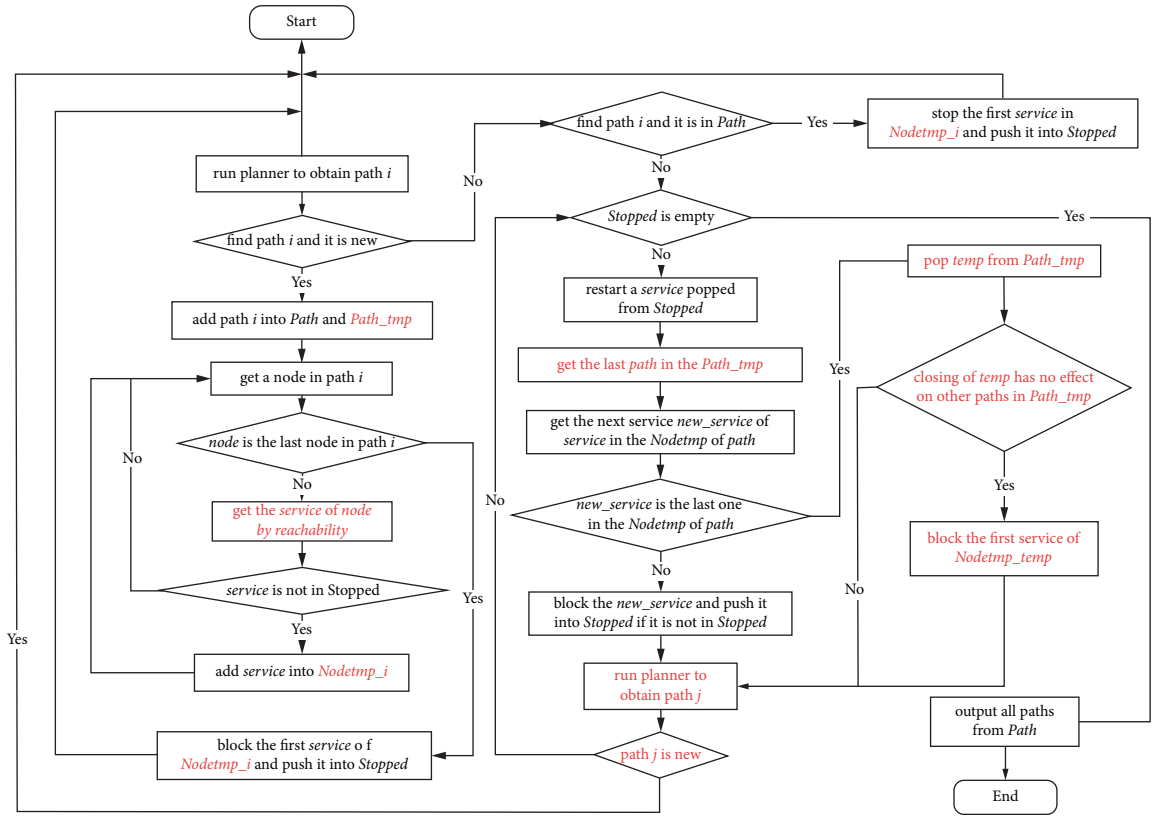

Figure 4: Flowchart for the improved attack path enumeration method (the improved parts are marked in red color).

block nodes on the attack path, which is implemented by commenting out some contents encoded in the file that relates to services.

In terms of the customized algorithm, a series of data structures are defined to be used in attack path enumeration. One of them, called *critical nodes*, denotes whether a node is always blocked to generate a new path or not. On the contrary, the other data structure, called *noncritical nodes*, denotes the nodes rely on some critical nodes, and no new paths are obtained if the nodes are blocked alone. Although these two data structures are applied for indicating how to block nodes, there exist fields of them to be predefined by analyzing the services in a manual way. It is not conducive to enumerate attack path in larger-scale networks. Hence, we remove these two data structures in our improved method. The modification of the attack path enumeration algorithm is shown as a flowchart in Figure 4.

```
    Input: device reachability graph Gr, subgraph size subg_size
    Output: all subgraphs
(1) function GENERATE SUBGRAPHS (Gr, subg_size)
(2)   if nodes_num of Gr more than subg_size then
(3)     push Gr to Qp
(4)     while Qp is not empty do
(5)       pop a subgraph G from Qp
(6)       find branch nodes Ns from G
(7)       get edges from Ns and push them into Qe
(8)       while Qe is not empty do
(9)         pop an edge qe from Qe
(10)        find successor subgraph Gs from edge qe in G
(11)        if nodes in Gs less than subg_size then
(12)          push Gs into Qp
(13)        else
(14)          push Gs into Qo
(15)    output all subgraphs from Qo
(16)  else
(17)    output Gr
(18) end function
```

Algorithm 1: Device reachability graph partitioning.

```
    Input: pddl file template domain_temp and problem_temp
      connection object to a graph database hg; planning goals
    Output: constructed pddl domain and problem files
(1) function GENERATE PDDL FILE (domain_temp, problem_temp, hg)
(2)   query device nodes, device reachability, vulnerability and component via hg
(3)   generate domain file:
(4)     generate predicates of vulnerability, reachability, pre and postconditions
(5)     generate actions of vulnerability from pre- and postconditions of vulnerability
(6)   end
(7)   generate problem file:
(8)     generate objects from device nodes
(9)     generate initially satisfied conditions
(10)    generate goals based on your input
(11)  end
(12)  return generated domain and problem files
(13) end function
```

Algorithm 2: Automatic construction of PDDL domain and problem files.

In our improved algorithm, we introduce two new data structures, named Nodetmp and Path_tmp, respectively, which are used to store the information during the enumeration. The Nodetmp is defined to record the services of the nodes in one path, and the Path_tmp is defined to record the path, which is not completely processed. The rest of the data structures are in accordance with the definitions in the literature [9].

5.4. Attack Graph Generation. In this part, we integrate the above three algorithms into a multithreading program. Each thread is assigned for one subgraph to call the subprogram of attack path enumeration, and creating and submitting a thread is managed by a thread pool. Particularly, we remove some irrelevant items for each subgraph from the problem PDDL file to reduce file parsing time. It greatly shortens the time to generate an attack graph in a large scale, and the result is discussed in Section 6.

The input is the number of threads. The output is a complete attack graph, merged by a set of attack graphs of subgraphs. After generating an attack graph, the exploit and attack edges among the nodes of devices and vulnerabilities are added in a graph database as mentioned in Table 2. More details are shown in the pseudocode of Algorithm 3.

6. Case Study

In this part, the proposed automatic planning-based attack path discovery approach is evaluated. At first, an

Input: number of threads thread_num
Output: attack graph *AG*; adding exploit and attack edges in a graph database
(1) create an empty attack graph *AG*
(2) get domain.pddl and problem.pddl via GENERATE PDDL FILE (domain_temp, problem_temp, *hg*)
(3) get all subgraphs from GENERATE SUBGRAPHS (*G*, subg_size)
(4) create threads pool threads_pool and set maxim workers corresponding to thread_num
(5) **foreach** subgraph in subgraphs **do**
(6) modify problem.pddl and domain.pddl based on *subgraph*
(7) create a thread and bind it to enumerate attack paths using a planner
(8) submit this thread to threads_pool
(9) **while** True **do**
(10) check the status of threads in threads_pool
(11) **if** all tasks in threads_pool have done **do**
(12) break
(13) generate subag from paths returned from each thread and merge them into *AG*
(14) get ag_edges from *AG*
(15) create attack and exploit edges in a graph database according to ag_edges

Algorithm 3: Attack graph generation in a multithreading manner.

experimental setup is introduced by a hypothetical network topology from IT to OT networks in Section 6.1. Then, attack paths are illustrated, and the corresponding data is stored in the form of graph data in Section 6.2. Finally, we discuss the performance in terms of device reachability graph partitioning, attack path planning, and its enumeration in Section 6.3, which allows us to examine the scalability with the increasing devices in the IT and OT networks.

6.1. Experimental Setup. As shown in Figure 5, a hypothetical network topology is constructed whose structure stems from the real-world practice, but its size is simplified. It is separated into six subnets according to the different functions. The Enterprise Control Network is a corporate network with respect to the product lifecycle management, the resource planning, the business planning, and so on. The Perimeter Network manages servers to provide information for users on the Enterprise Control Network via a variety of services, such as web and mail. The Manufacturing Operations Network is a bridge of information exchanges between control systems and enterprise resources planning systems to support the top-down decision-making. The Process Control Network is used to transmit instructions and data between control and measurement units and Supervisory Control and Data Acquisition (SCADA) devices. The Automatic Control Network is connected to numbers of HMIs and PLCs in fields, which are responsible for logic and control computing tasks to manipulate and regulate sensors or actuators in the Physical Control Network. Among them, the IT networks are made up of the Enterprise Control Network and the Perimeter Network, and other subnets belong to the OT networks [1].

The hypothetical network topology contains twenty heterogeneous devices so as to introduce many services and vulnerabilities, as shown in Tables 3 and 4. The vulnerability information is extracted from descriptions of the NVD (https://nvd.nist.gov/) and the ATT & CK ICS (https://collaborate.mitre.org/attackics/index.php/Main_Page). Considering the device type and applied technologies, we divide the whole network topology into two parts, namely, Zones A and B. In Zone A, more devices adopt the commercial-off-the-shelf software and hardware, where more vulnerabilities may be exploited for the purpose of lateral movements to the OT networks. Due to factors, such as time and continuity, less security protection devices are deployed in Zone B. Once some devices are compromised in that zone, sophisticated attackers can take multiple measures to launch control process-oriented attacks to affect physical operations. In order to highlight dependencies of vulnerabilities, we define access control rules among services in detail, as shown in Table 5.

6.2. Attack Path Discovery. Based on the model constructed in Section 4.3, we store the experimental data in the form of the graph data. Utilized in this paper, the graph database, HugeGraph (https://hugegraph.github.io/hugegraph-doc/), is efficient, universal, and open source. It is fully compatible with Gremlin query language and implements with the Apache TinkerPop3 framework. The stored graph data is the basis of subsequent automatic generation of the PDDL files and the final attack graph generation. To demonstrate the feasibility of our proposed method, we give the results in a reversed order as it is described in Section 3. In this experiment, we define the entry point of the attack as the Manger PC, and its compromised goal is the PLC2 that is a slave station connected to a set of physical equipment.

Figure 6 is a complete attack graph for the experimental environment, which is output by the Graphviz (http://www.graphviz.org/) library of the Python. There are 189 attack paths that can reach the attack goal. We separately show the attack paths for Zones A and B in Tables 6 and 7, because of display convenience. Obviously, it is difficult to find a node like the Historian node (Dev12) of the hypothetical network topology, which can be viewed as a cut point in the Graph Theory to partition a network topology.

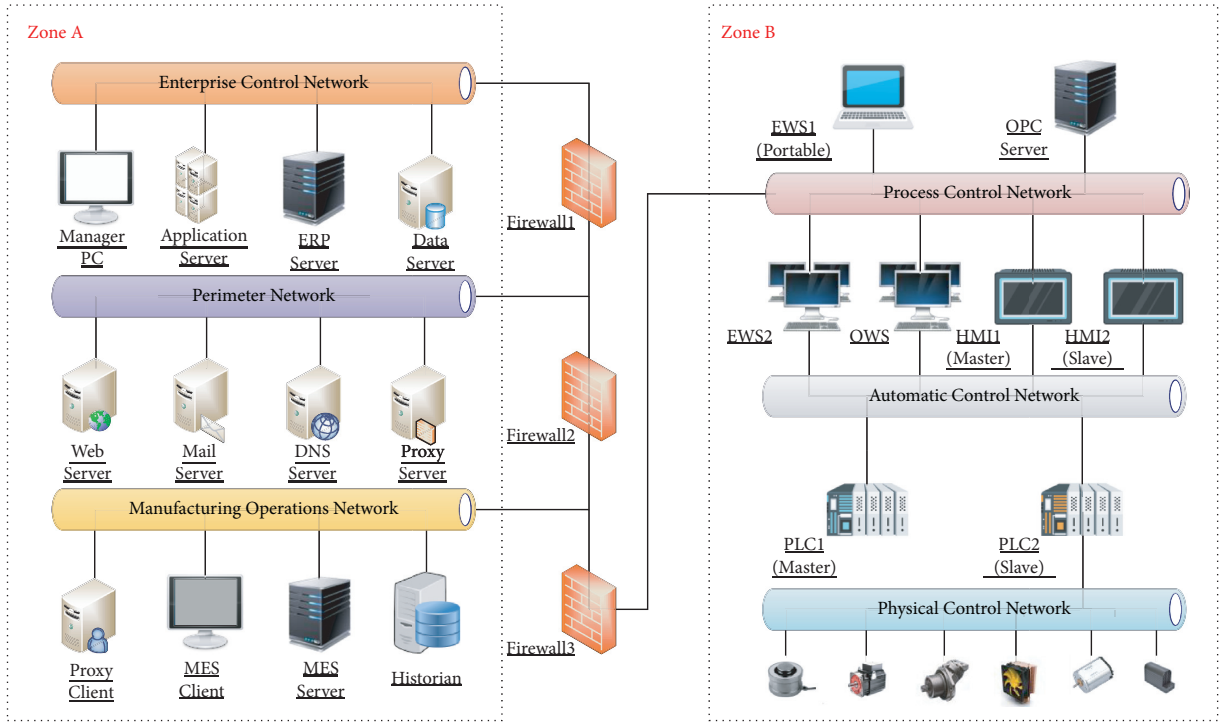

Figure 5: A hypothetical network topology from IT to OT networks.

Table 3: Device information in the experimental environment.

Dev.ID	Device name	Port	Vulnerability	Affected component
Dev1	Manager PC	—	LNK remote code execution	Icon of the shortcut in windows platform
		—	Credentials leak	Connected device login
Dev2	Application server	22	OS command injection	OpenSSH(SCP)
Dev3	ERP server	3389	BITS improper privilege management	Windows background intelligent transfer service
Dev4	Data server	3306	Permissions and access controls	MySQL
Dev5	Web server	80	Memory buffer overflow	Internet information services
Dev6	Mail server	80	Improper access control	Roundcube
Dev7	DNS server	53	DNS server remote code execution	Windows DNS server
Dev8	Proxy server	8090, 4900	Path traversal	Lanproxy server
Dev9	Proxy client	12000	Plaintext credential	Lanproxy client
Dev10	MES client	445	SMBv3 remote code execution	Microsoft server message block protocol
		—	Credentials leak	Connected device login
Dev11	MES server	22	Kernel improper privilege management	Linux kernel
Dev12	Historian	80	SQL server remote code execution	Microsoft SQL server reporting services
Dev13	EWS1	445, 139	Code injection	MSRPC over SMB
Dev14	EWS2	3389	Brute force	Remote desktop services
Dev15	OWS	445	SMB remote code execution	Microsoft server message block protocol
Dev16	OPC server	8080	Unrestricted upload of file	Apache tomcat
Dev17	HMI1 (master)	2308, 1033	Modify configuration project	HMI configuration project in WinCC
Dev18	HMI2 (slave)	2308, 1034	Modify configuration project	HMI Configuration project in WinCC
			Fake MAC address	HMI and PLC communication
Dev19	PLC1 (master)	102	Modify parameters/modes	PLC automatic operation/states
			Modify control logic	PLC program project in TIA portal
			Plaintext control command	Legacy S7Comm protocol
Dev20	PLC2 (slave)	102, 502	Fake MAC address	HMI and PLC communication
			Modify parameters/modes	PLC automatic operation/states
			Modify control logic	PLC program project in TIA portal
			Plaintext control command	Modbus protocol
			Uncontrolled resource consumption	Protocol common used port
			Improper control	CPU defect mode

Table 4: Vulnerability information in the experimental environment.

Vul.ID	Vulnerability	ATT_Vector	Precondition	Postcondition	Tactics/techniques
Vul1	LNK remote code execution	Local	USB access/crafted LNK files	Execute any code	IA/replication through removable media
Vul2	Credentials leak	Local	Plaintext record file	Credential acquisition	LM/valid accounts
Vul3	OS command injection	Remote	SSH password	Execute any code	LM/remote services
Vul4	BITS improper privilege management	Local	USER login	Administrator (windows)	PE/exploitation for privilege escalation
Vul5	Permissions and access controls	Remote	USER login	Root (linux)	PE/exploitation for privilege escalation
Vul6	Memory buffer overflow	Remote	Crafted URL	Execute any code	IA/exploit public-facing application
Vul7	Improper access control	Remote	Crafted e-mail messages	Execute any code	IA/exploit public-facing application
Vul8	DNS server remote code execution	Remote	Malicious requests	Execute any code	IA/exploit public-facing application
Vul9	Path traversal	Remote	Port scan	Credential acquisition	CA/exploitation for credential access
Vul10	Plaintext credentials	Remote	Credential	Login	LM/valid accounts
Vul11	SMBv3 remote code execution	Remote	USER	Execute any code	LM/exploitation of remote services
Vul12	Kernel improper privilege management	Local	USER login	Root (linux)	PE/exploitation for privilege escalation
Vul13	SQL server remote code execution	Remote	Incorrect page request	Execute any code	IA/exploit public-facing application
Vul14	Code injection	Remote	Crafted RPC request	Execute any code	LM/exploitation of remote services
Vul15	Brute force	Remote	Credential	Login	LM/valid accounts
Vul16	SMB remote code execution	Remote	USER	Execute any code	LM/exploitation of remote services
Vul17	Unrestricted upload of file	Remote	JSP file/HTTP request	Execute any code	IA/exploit public-facing application
Vul18	Modify Configuration project	Remote	Malicious Configuration project	Impair HMI control function	P/modify program
Vul19	Modify control logic	Remote	Malicious control logic	PLC denial of service	P/modify program
Vul20	Modify parameters/Modes	Remote	Malicious operations	PLC denial of service	IPC/modify parameter
Vul21	Plaintext control command	Remote	Crafted control command	PLC denial of service	IPC/unauthorized command message
Vul22	Uncontrolled resource consumption	Remote	High volume of requests	PLC denial of service	IRF/denial of service
Vul23	Fake MAC address	Remote	Scan devices/traffic forward/ Modify data	PLC denial of service	C/man in the middle
Vul24	Improper control	Remote	Crafted packets	PLC denial of service	E/change operating mode

The paths in the attack graph are a mix of the Device nodes and the Vulnerability nodes, which represents that an attacker can reach a device and exploit vulnerability. As discussed in Section 3, attack graph generated by our proposed approach is to analyze exploitation dependencies, and the corresponding edges are added among nodes in the graph database. Afterward, it is convenient to find attack paths arbitrarily with a fixed entry point and an attacking goal by the Gremlin query. In that way, attack paths in Table 6 are listed from the Attacker to the Historian device. Similarly, attack paths in Table 7 are listed from the Historian device to the PLC2.

6.3. Performance Evaluation. Performance evaluation tests of our proposed approach are carried out in the following environments. The domain and problem of the hypothetical network topology are described in the PDDL (Version 2.2) (https://planning.wiki/ref/pddl22). The attack path planning is executed on the SGPlan (Version 5.2.2). All programs on the four algorithms mentioned in Section 5 are implemented in Python (Version 3.5.2), running on a Linux server with the Intel Xeon Silver 4110 CPU at 2.1 GHz and 125 GB RAM. HugeGraph (Server Version 0.11.2) runs standalone in a Docker (Version 19.03.12) container.

Initially, we implement the method introduced in the literature [9] to validate its scalability. Assuming that a one-to-one correspondence exists between the number of vulnerabilities and the number of devices, changes in the network topology are only reflected in the complexity of the problem file in PDDL that has a great impact on the planning. The results are shown in Table 8. Column 2 on the left concerns the size of each test network topology, while the remaining columns, respectively, fucus on the performance

Table 5: Device access control rules in the experimental environment.

Dom.ID	Domain name	Source device	Destination devices
Dom1	Internet	Manager PC	(Application server, 22) (ERP server, 3389) (web server, 80) (proxy server, 8090&4900)
		Application server	(Data server, 3306)
		ERP server	(Data server, 3306)
		Data server	(Mail server, 80) (DNS server, 53)
Dom2	DMZ	Web server	(Historian, 80)
		Mail server	(Proxy server, 8090 & 4900)
		DNS server	(Proxy server, 8090 & 4900)
		Proxy server	(Proxy client, 1200)
Dom3	Scheduling	Proxy client	(Historian, 80) (MES client, 445)
		MES client	(MES server, 22)
		MES server	(Historian, 80)
		Historian	(OPC server, 8080)
Dom4	Supervision	OPC server	(EWS1, 445 & 139) (EWS2, 3389) (OWS, 445)
		EWS1	(HMI1, 2308 & 1033) (HMI2, 2308, & 1033)
		EWS2	(PLC1, 102) (PLC2, 102 & 502)
		OWS	(HMI2, 2308 & 1033) (PLC1, 102) (PLC2, 102 & 502)
Dom5	Process	HMI1	(PLC1, 102)
		HMI2	(PLC2, 102)
		PLC1	(PLC2, 502)

surrounding the CPU time and memory, including the total planning time in the SGPlan, the parsing time for PDDL files, the enumeration time for all attack paths, and the memory consumption. With the increase of devices (nodes) in the network topology, it is clear that both the running time and the memory consumption grow exponentially. The SGPlan provides the planning time and the parsing time for PDDL files as outputs, and the planning time involves the parsing time and pure solution time for each valid attack path. By summing the planning time and the parsing time, we find that the parsing time accounting for more than 90% dominates in the planning time with the complexity of the problem file in the PDDL growing. Moreover, it is worth noting that the enumeration time is even more unsatisfactory for a small network topology, taking nearly 2 hours to generate all attack paths for a network topology with a size of 41 devices.

To overcome those two shortcomings in the attack path enumeration method using the planner while applying in the large-scale networks, we introduce the graph partition algorithm into our proposed approach. There is a key parameter, namely, the subgraph size subg_size, which is set to 10 in Algorithm 1. Performance results of each stage are shown in Table 9. From row 2 to 4 in the table, the results are better than those shown in Table 8 in the case of 21 topology nodes. The reason why the time-consuming process of planning and parsing drastically decreases is that partitioning algorithm on the basis of branch points reduces the complexity of each problem file in PDDL, which helps make it easier for the planner to parse the file and solve a solution for each subgraph. In addition, we further provide running time in the graph partitioning (row 6) and operations with the HugeGraph, such as importing and traversal query (rows 7 and 8), and they take a small proportion in the running time of our proposed approach. The remaining rows list the information on the attack graph of the hypothetical network topology in Section 6.1.

Finally, we validate that each stage of our proposed approach is suitable for a large-scale network by the experiment. The hypothetical network topology with increasing size of devices and complexity are considered and discussed. It simply achieves that goal by integrally replicating several times those devices contained in the topology to build scenarios of different network sizes, the number of devices ranging from 100 to 1000. Experimental results are shown from Figures 7 to 10. In Figure 7, running time and memory usage of device reachability graph partitioning are shown, considering different numbers of devices. It is observed that the running time is less than 0.2 seconds, even though topological scale reaches more than 1000 devices. Figures 8 and 9, respectively, show the comparison between multithreading and single-threading modes in running time and the memory usage. We set the thread number of the threading pool in Algorithm 3 as 20. Apparently, the time consumption in the multithreading way is less than that in the single-threading way. Simultaneously, its growth ratio is also slower with the increasing of devices. But the cost of our proposed approach has higher memory usage than that in the single-threading way, which is almost linear growth. The overhead of multithreading exists in parsing the problem files and enumerating attack paths. Hence, we remove the irrelevant content encoded in the problem files based on the devices of each subgraph to reduce time of parsing and planning in each thread. It makes it possible to avoid blocking or restarting invalid services in attack path enumeration algorithm as well. What is more, the efficiency of operations, such as importing data and traversal query, determines the process of automatic constructing PDDL files and searching attack paths. Figure 10 shows the running time of the two key operations of the HugeGraph in Algorithms 2 and 3, considering different numbers of devices.

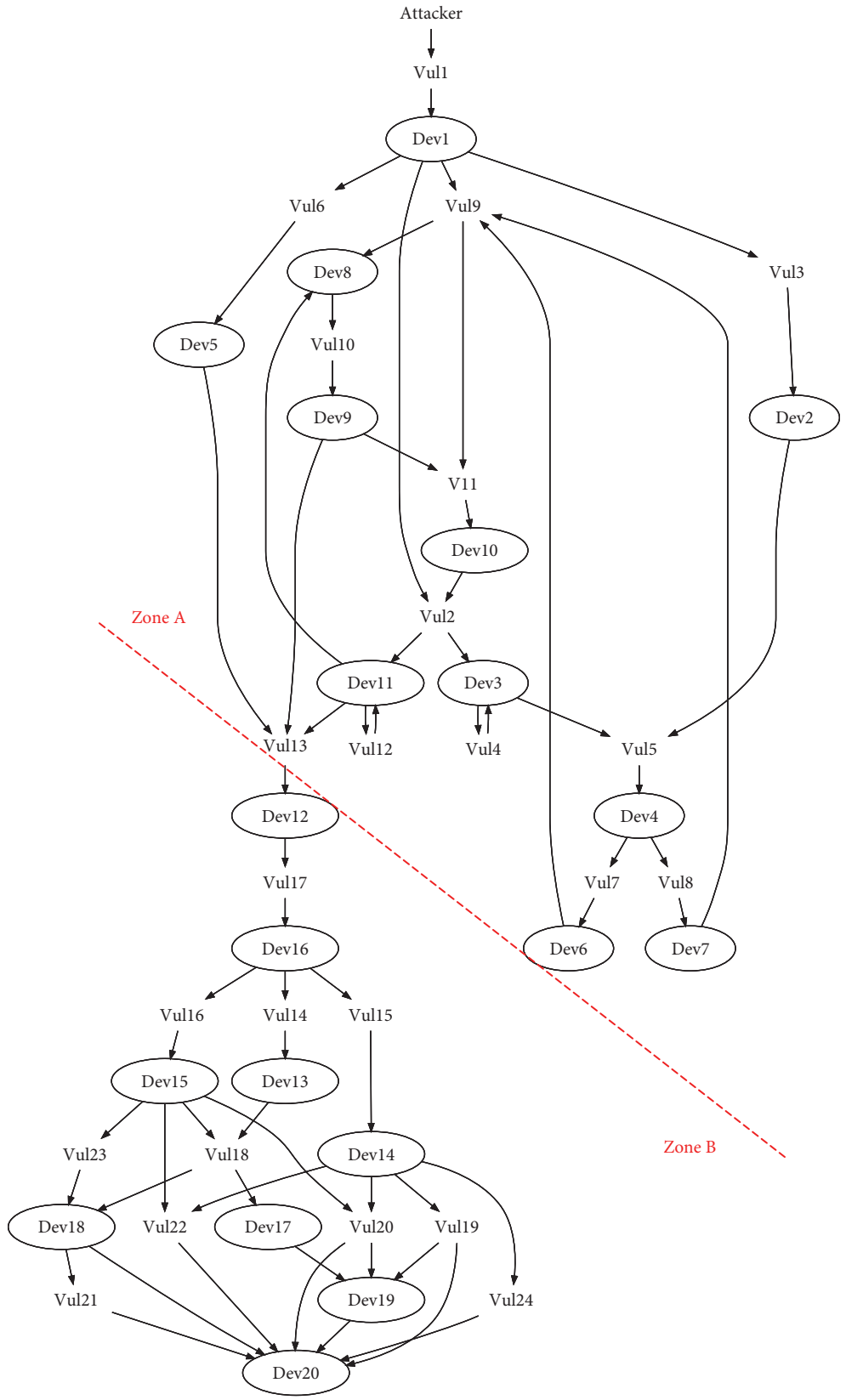

Figure 6: Attack graph of the experimental environment.

Table 6: Attack paths in Zone A.

No.	Attack paths
1	Attacker⟶Vul1⟶Dev1⟶Vul6⟶Dev5⟶Vul13⟶Dev12
2	Attacker⟶Vul1⟶Dev1⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶Vul13⟶Dev12
3	Attacker⟶Vul1⟶Dev1⟶Vul3⟶Dev2⟶Vul5⟶Dev4⟶Vul7⟶Dev6⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶Vul13⟶Dev12
4	Attacker⟶Vul1⟶Dev1⟶Vul3⟶Dev2⟶Vul5⟶Dev4⟶Vul8⟶Dev7⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶Vul13⟶Dev12
5	Attacker⟶Vul1⟶Dev1⟶Vul2⟶Dev3⟶Vul4⟶Dev3⟶Vul5⟶Dev4⟶Vul7⟶Dev6⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶Vul13⟶Dev12
6	Attacker⟶Vul1⟶Dev1⟶Vul2⟶Dev3⟶Vul4⟶Dev3⟶Vul5⟶Dev4⟶Vul8⟶Dev7⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶Vul13⟶Dev12
7	Attacker⟶Vul1⟶Dev1⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶V11⟶Dev10⟶Vul2⟶Dev11⟶Vul12⟶Dev11⟶Vul13⟶Dev12
8	Attacker⟶Vul1⟶Dev1⟶Vul3⟶Dev2⟶Vul5⟶Dev4⟶Vul7⟶Dev6⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶V11⟶Dev10⟶Vul2⟶Dev11⟶Vul12⟶Dev11⟶Vul13⟶Dev12
9	Attacker⟶Vul1⟶Dev1⟶Vul3⟶Dev2⟶Vul5⟶Dev4⟶Vul8⟶Dev7⟶Vul9⟶V11⟶Dev10⟶Vul2⟶Dev11⟶Vul12⟶Dev11⟶Dev8⟶Vul10⟶Dev9⟶Vul13⟶Dev12
10	Attacker⟶Vul1⟶Dev1⟶Vul2⟶Dev3⟶Vul4⟶Dev3⟶Vul5⟶Dev4⟶Vul7⟶Dev6⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶V11⟶Dev10⟶Vul2⟶Dev11⟶Vul12⟶Dev11⟶Vul13⟶Dev12
11	Attacker⟶Vul1⟶Dev1⟶Vul2⟶Dev3⟶Vul4⟶Dev3⟶Vul5⟶Dev4⟶Vul8⟶Dev7⟶Vul9⟶Dev8⟶Vul10⟶Dev9⟶V11⟶Dev10⟶Vul2⟶Dev11⟶Vul12⟶Dev11⟶Vul13⟶Dev12

Table 7: Attack paths in Zone B.

No.	Attack paths
1	Dev12⟶Vul17⟶Dev16⟶Vul14⟶Dev13⟶Vul18⟶Dev18⟶Dev20
2	Dev12⟶Vul17⟶Dev16⟶Vul14⟶Dev13⟶Vul18⟶Dev17⟶Dev19⟶Dev20
3	Dev12⟶Vul17⟶Dev16⟶Vul15⟶Dev14⟶Vul19⟶Dev20
4	Dev12⟶Vul17⟶Dev16⟶Vul15⟶Dev14⟶Vul19⟶Dev19⟶Dev20
5	Dev12⟶Vul17⟶Dev16⟶Vul15⟶Dev14⟶Vul22⟶Dev20
6	Dev12⟶Vul17⟶Dev16⟶Vul15⟶Dev14⟶Vul24⟶Dev20
7	Dev12⟶Vul17⟶Dev16⟶Vul15⟶Dev14⟶Vul20⟶Dev20
8	Dev12⟶Vul17⟶Dev16⟶Vul15⟶Dev14⟶Vul20⟶Dev19⟶Dev20
9	Dev12⟶Vul17⟶Dev16⟶Vul16⟶Dev15⟶Vul20⟶Dev20
10	Dev12⟶Vul17⟶Dev16⟶Vul16⟶Dev15⟶Vul20⟶Dev19⟶Dev20
11	Dev12⟶Vul17⟶Dev16⟶Vul16⟶Dev15⟶Vul22⟶Dev20
12	Dev12⟶Vul17⟶Dev16⟶Vul16⟶Dev15⟶Vul18⟶Dev18⟶Dev20
13	Dev12⟶Vul17⟶Dev16⟶Vul16⟶Dev15⟶Vul23⟶Dev18⟶Vul21⟶Dev20

Table 8: Performances of attack path enumeration in [9].

No.	Topo_nodes	Plan_time (s)	Parse_time (s)	Enum_time (s)	Memory (KB)
1	11	2.23	1.66	14.72	39256
2	21	44	40.96	184.49	164360
3	31	363.88	354.19	1398.11	644428
4	41	1954.15	1928.19	7319.67	1857944
5	51	7033.4	6973.02	26486.55	4524044

Table 9: Performances of our proposed method.

No.	Metrics	Statistics
1	Topo_nodes	21
2	Plan_time (s)	0.11
3	Parse_time (s)	0.1
4	Enum_time (s)	53.94
5	Memory (KB)	19292
6	partition_time (s)	0.019
7	import_time (s)	0.27
8	tranversal_time (s)	0.007
9	ag_nodes	43
10	ag_edges	63
11	att_paths	189

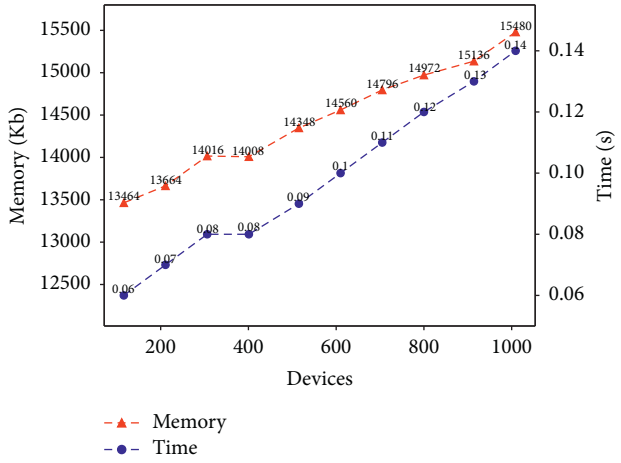

Figure 7: Running time and memory usage considering different numbers of devices in Algorithm 1.

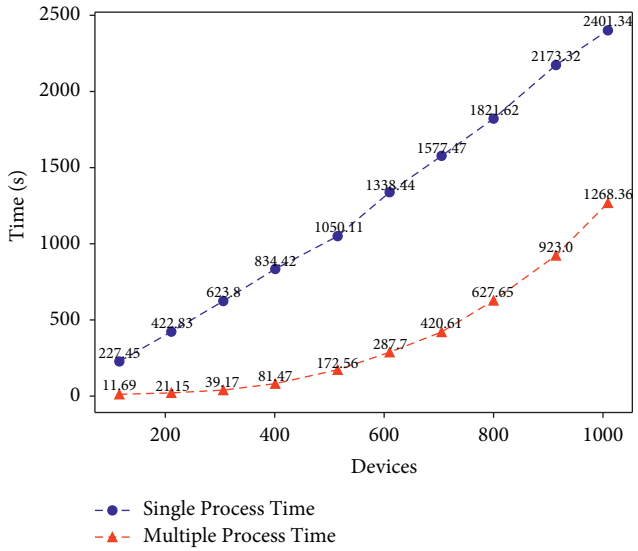

Figure 8: Running time comparison of the single and the proposed multiple threading method considering different numbers of devices.

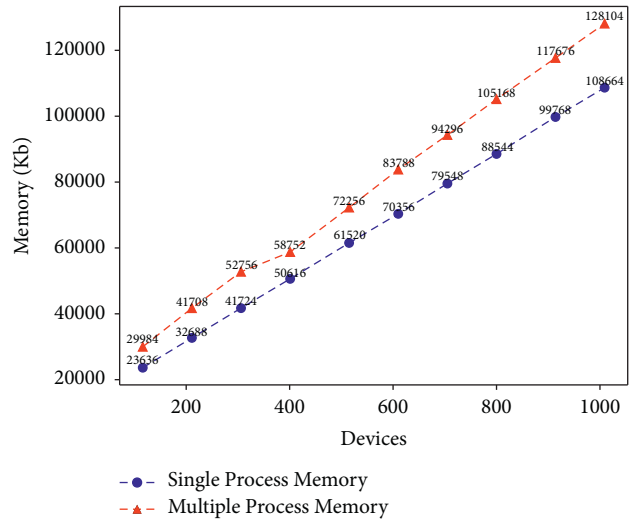

Figure 9: Memory usage comparison of the single and the proposed multiple threading method considering different numbers of devices.

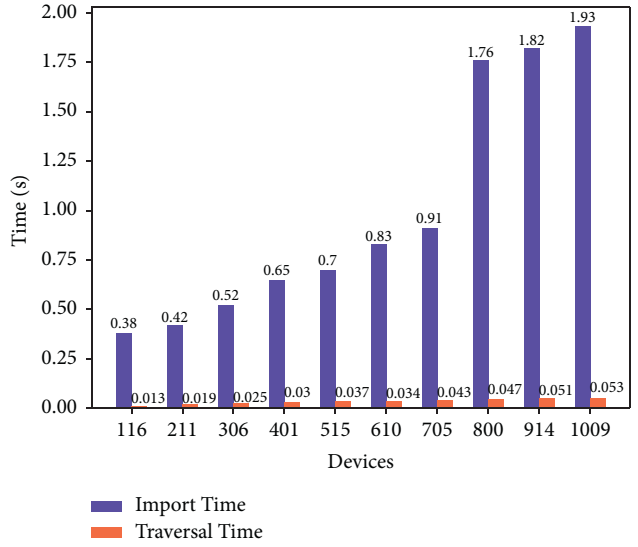

Figure 10: Running time of importing and traversal query of the HugeGraph considering different numbers of devices.

7. Conclusion

There is an overwhelming trend that IT and OT networks appear to be deeply integrated into the current industry, whereas their existing security issues still cannot be ignored. Concentrating on attack paths, in contrast with a standalone vulnerability scanning, security assessment takes consideration of dependencies among devices, vulnerabilities, and networks as a whole. Focusing on all attack paths terminated to the specific goal, we present an automatic planning-based approach for the attack graph generation. The conventional planning-based attack path discovery approach is improved with the graph data management, topology partitioning, and the parallel execution, adapted to the large-scale IT and OT networks. The formal data representation adopts the PDDL for describing attack scenarios, which still possesses the advantages in the modeling. Meanwhile, multisource and scattered data is managed by a graph database, providing opportunities for users to query attack paths and corresponding information.

Experimental results indicate that our proposed approach manifests improvements in automation and scalability compared with the conventional planning-based method. Device reachability graph partitioning algorithm helps to reduce the time consuming of parsing the PDDL problem file and planning a single attack path. Calling the attack path enumeration in a multithreading manner has more desirable performance with the number of devices growing. Using the graph database like HugeGraph guarantees the efficiency of importing data and traversal query, which does favor to complete tasks, such as automatic construction of the PDDL files and search attack paths.

In the future work, we attempt to utilize variety of domain-independent AI planners to discovery attack paths, but it is not limited to finding the shortest path in each iteration. The research direction will shift to quantitative security assessment to analyze the attack paths, integrating with probabilities of critical nodes. To further predict attack behaviors, we introduce logical reasoning and the uncertainty theory into the graph data model. Additionally, visual optimization of attack paths is worthy to implement for large-scale IT and OT networks.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

All the authors hereby declare no conflicts of interest.

Acknowledgments

This research was funded by the National Key Research and Development Program of China (no. 2018YFB2004200).

References

[1] R. Paes, D. C. Mazur, B. K. Venne, and J. Ostrzenski, "A guide to securing industrial control networks: integrating IT and OT systems," *IEEE Industry Applications Magazine*, vol. 26, no. 2, pp. 47–53, 2019.

[2] T. Alladi, V. Chamola, and S. Zeadally, "Industrial control systems: cyberattack trends and countermeasures," *Computer Communications*, vol. 155, pp. 1–8, 2020.

[3] A. Volkova, M. Niedermeier, R. Basmadjian, and H. D. Meer, "Security challenges in control network protocols: a survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 619–639, 2018.

[4] R. J. Thomas and T. Chothia, "Learning from vulnerabilities–categorising, understanding and detecting weaknesses in industrial control systems," *Computers & Security*, pp. 100–116, 2020.

[5] D. Cerotti, D. Codetta-Raiteri, G. Dondossola et al., "A bayesian network approach for the interpretation of cyber attacks to power systems," in *Proceedings of the Italian Conference on CyberSecurity*, Pisa, Italy, February 2019.

[6] H. S. Lallie, K. Debattista, and J. Bal, "A review of attack graph and attack tree visual syntax in cyber security," *Computer Science Review*, vol. 35, Article ID 100219, 2020.

[7] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz, and J. Lopez, "A survey of iot-enabled cyberattacks: assessing attack paths to critical infrastructures and services," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3453–3495, 2018.

[8] J. Zeng, I. Wu, I. Yanyu Chen, I. Rui Zeng, and I. Chengrong Wu, "Survey of attack graph analysis methods from the perspective of data and knowledge processing," *Security and Communication Networks*, vol. 2019, Article ID 2031063, 16 pages, 2019.

[9] N. Ghosh and S. K. Ghosh, "A planner-based approach to generate and analyze minimal attack graph," *Applied Intelligence*, vol. 36, no. 2, pp. 369–390, 2012.

[10] S. Khan and S. Parkinson, "Review into state of the art of vulnerability assessment using artificial intelligence," *Guide to Vulnerability Analysis for Computer Networks and Systems*, Springer, Heidelberg, Germany, pp. 3–32, 2018.

[11] D. R. McKinnel, T. Dargahi, A. Dehghantanha, and K.-K. R. Choo, "A systematic literature review and meta-analysis on artificial intelligence in penetration testing and vulnerability assessment," *Computers & Electrical Engineering*, vol. 75, pp. 175–188, 2019.

[12] T. Alaa, "A2G2V: automatic attack graph generation and visualization and its applications to computer and SCADA networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, pp. 3488–3498, 2019.

[13] M. Cheminod, L. Durante, L. Seno, and A. Valenzano, "Detection of attacks based on known vulnerabilities in industrial networked systems," *Journal of Information Security and Applications*, vol. 34, pp. 153–165, 2017.

[14] S. Wu, Y. Zhang, and W. Cao, "Network security assessment using a semantic reasoning and graph based approach," *Computers & Electrical Engineering*, vol. 64, pp. 96–109, 2017.

[15] M. Inokuchi, Y. Ohta, S. Kinoshita et al., "Design procedure of knowledge base for practical attack graph generation," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, Auckland New Zealand, July 2019.

[16] O. Stan, R. Bitton, M. Ezrets et al., "Extending attack graphs to represent cyber-attacks in communication protocols and modern IT networks," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[17] T. Gonda, R. Puzis, and B. Shapira, "Scalable attack path finding for increased security," in *Proceedings of the*

International Conference on Cyber Security Cryptography and Machine Learning, July 2017.

[18] B. Bezawada, I. Ray, and K. Tiwary, "AGBuilder: an AI tool for automated attack graph building, analysis, and refinement," in *Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy*, July 2019.

[19] R. Mirsky, Y. Shalom, A. Majadly, Y. Gal, R. Puzis, and A. Felner, "New goal recognition algorithms using attack graphs," in *Proceedings of the International Symposium on Cyber Security Cryptography and Machine Learning*, June 2019.

[20] Z. Yichao, Z. Tianyang, G. Xiaoyue, and W. Qingxian, "An improved attack path discovery algorithm through compact graph planning," *IEEE Access*, vol. 7, pp. 59346–59356, 2019.

[21] K. Kaynar and F. Sivrikaya, "Distributed attack graph generation," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 519–532, 2015.

[22] H. Li, Y. Wang, and Y. Cao, "Searching forward complete attack graph generation algorithm based on hypergraph partitioning," *Procedia Computer Science*, vol. 107, pp. 27–38, 2017.

[23] N. Cao, K. Lv, and C. Hu, "An attack graph generation method based on parallel computing," in *Proceedings of the International Conference on Science of Cyber Security*, August 2018.

[24] Y. Chen, Z. Liu, Y. Liu, and C. Dong, "Distributed attack modeling approach based on process mining and graph segmentation," *Entropy*, vol. 22, no. 9, Article ID 1026, 2020.

[25] J. Lee, D. Moon, I. Kim, and Y. Lee, "A semantic approach to improving machine readability of a large-scale attack graph," *The Journal of Supercomputing*, vol. 75, no. 6, pp. 3028–3045, 2019.

[26] X. Mao, M. Ekstedt, E. Ling, E. Ringdahl, and L. Robert, "Conceptual abstraction of attack graphs-A use case of securiCAD," in *Proceedings of the International Workshop on Graphical Models for Security*, June 2019.

[27] M. S. Barik and C. Mazumdar, "A graph data model for attack graph generation and analysis," in *Proceedings of the International Conference on Security in Computer Networks and Distributed Systems*, March 2014.

[28] S. Noel, E. Harley, K. H. Tam, M. Limiero, and M. Share, "CyGraph," *Handbook of Statistics*, Elsevier, vol. 35, pp. 117–167, Amsterdam, Netherlands, 2016.

[29] N. Polatidis, M. Pavlidis, and H. Mouratidis, "Cyber-attack path discovery in a dynamic supply chain maritime risk management system," *Computer Standards & Interfaces*, vol. 56, pp. 74–82, 2018.

[30] B. Yuan, Z. Pan, F. Shi, and Z. Li, "An attack path generation methods based on graph database," in *Proceedings of the 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, June 2020.