Karbala International Journal of Modern Science

Volume 8 | Issue 3 Article 3

Attack Prediction to Enhance Attack Path Discovery Using Improved Attack Graph

Zaid. J. Al-Araji
Universiti Teknikal Malaysia Melaka, Melaka, Malaysia, zaid.jassim4@gmail.com

Sharifah Sakinah Syed Ahmad
Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

Raihana Syahirah Abdullah
Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

Recommended Citation
Al-Araji, Zaid. J.; Syed Ahmad, Sharifah Sakinah; and Abdullah, Raihana Syahirah (2022) "Attack Prediction to Enhance Attack Path Discovery Using Improved Attack Graph," *Karbala International Journal of Modern Science*: Vol. 8 : Iss. 3 , Article 3.
Available at: https://doi.org/10.33640/2405-609X.3235

Attack Prediction to Enhance Attack Path Discovery Using Improved Attack Graph

Abstract

Organisations and governments constantly face potential security attacks. However, the need for next-generation cyber defence has become even more urgent in a day and age when attack surfaces that hackers can exploit have grown at an alarming rate with an increase in the number of devices that are connected to the Internet. As such, next-generation cyber defence that relies on predictive analysis is more proactive than existing technologies that rely on intrusion detection. Many approaches with which to detect and predict attacks have been proposed in recent times. One such approach is attack graphs. The primary purpose of an attack graph is to not only predict an attack but its next steps within a network as well as. More specifically, an attack graph depicts the paths that an attacker may employ to circumvent network policies by exploiting interdependencies between the vulnerabilities. However, extant attack graphs are plagued with a few issues. Scalability is just one of the main issues that attack graph generation faces. This is because an increase in the number of devices used increases the number of vulnerabilities within a network. This, in turn, increases the complexity as well as the amount of time required to generate an attack graph. At present, existing studies that have used attack graphs to predict the subsequent steps during an attack have had to manually assigned the attack location for attack graph analysis. In order to overcome this limitation, this present study recommends the use of intelligent agents to reduce reachability time by calculating between the nodes as well as using the A* prune algorithm to remove useless edges and reduce attack graph complexity. For the attack graph analysis, the random forest (RF) algorithm was used to detect, predict, and dynamically ascertain the attack location in the network. The results of the attack graph generation experiment revealed that the A* prune attack graph produced better results than existing attack graphs.

Keywords

Attack Graph, Attack Path, A* prune algorithm, attack path discovery, attack graph analysis

RESEARCH PAPER

Attack Prediction to Enhance Attack Path Discovery Using Improved Attack Graph

Zaid J. Al-Araji [a,b,*], Sharifah Sakinah Syed Ahmad [a], Raihana Syahirah Abdullah [a]

[a] Universiti Teknikal Malaysia Melaka, Melaka, Malaysia
[b] University of Mosul, Mosul, Iraq

Abstract

Organizations and governments constantly face potential security attacks. However, the need for next-generation cyber defense has become even more urgent in a day and age when attack surfaces that hackers can exploit have grown at an alarming rate with an increase in the number of connected devices to the Internet. The next-generation cyber defense that relies on predictive analysis is more proactive than existing technologies that rely on intrusion detection. Many approaches with which to detect and predict attacks have been proposed in recent times. One such approach is attack graphs. The primary purpose of an attack graph is to not only predict an attack but its next steps within a network as well. More specifically, an attack graph depicts the paths that an attacker may employ to circumvent network policies by exploiting interdependencies between the vulnerabilities. However, extant attack graphs are plagued with a few issues. Scalability is just one of the main issues that attack graph generation faces. This is because an increase in the number of devices used increases the number of vulnerabilities within a network. This, in turn, increases the complexity as well as the amount of time required to generate an attack graph. At present, existing studies that have used attack graphs to predict the subsequent steps during an attack have manually assigned the attack location for attack graph analysis. In order to overcome this limitation, this present study recommends the use of intelligent agents to reduce reachability time by calculating between the nodes, as well as using the A* prune algorithm to remove useless edges and reduce attack graph complexity. For the attack graph analysis, the random forest algorithm was used to detect, predict, and dynamically ascertain the attack location in the network. The results of the attack graph generation experiment revealed that the A* prune attack graph produced better results than existing attack graphs.

Keywords: Attack graph, Attack path, A* prune algorithm, Attack path discovery, Attack graph analysis

1. Introduction

The exponential growth of computer networking technologies has considerably changed the way that people live [1]. Networks have permeated every aspect of life and break the limits of space and time for the benefit of humanity. According to Statista [2], the number of devices connected through the Internet increases annually. In 2010, there were around eight billion devices connected through the Internet. This number increased by 20% to 10 billion in 2021.

The exponential growth of Internet interconnections increases security concerns [3]. Furthermore, the increase in the usage of devices through the Internet also increases vulnerabilities within a network, which results in increased attacks on organisations and individual networks [4]. Amidst Malaysia's significant commitment to cyber security and its ranking as the third most secure country globally, there were 6274 cyberattacks reported in 2017 [5]. This indicates that cyberspace cannot be entirely secure. Consequently, cyber security is becoming increasingly important as the world's reliance on information technology and the Internet grows. These attacks cause tremendous damage to individuals and organisations alike.

As the cybersecurity community accepted that they could not entirely eliminate cyber-attacks, the

Received 28 January 2022; revised 5 May 2022; accepted 9 May 2022.
Available online 1 August 2022

* Corresponding author at: Universiti Teknikal Malaysia Melaka, Melaka, Malaysia.
E-mail address: zaid.jassim4@gmail.com (Z.J. Al-Araji).

https://doi.org/10.33640/2405-609X.3235

focus of current studies shifted to prevention, detection, prediction, and lowering the impact of security incidents [6]. Many approaches that prevent, detect, and predict attacks have been proposed [7]. One of these approaches is attack graphs.

Phillips and Swiler proposed attack graphs in 1998 [8]. Since then, several researchers have used various methods of generating attack graphs in order to enhance the attack graph model [9]. An attack graph is a very useful tool that provides an abstract depiction of all the possible paths that attackers may use to compromise a network [10]. It comprises vertices and directed edges, with vertices representing network states and edges representing state transitions. Attack graph construction combines vulnerabilities with access control rules to show all possible attack paths within a network, from source to target. There are two ways to generate an attack graph which are using a tool like [11] use MalVAL tool, or using program languages. Using the program languages, there are four stages to generate an attack graph; reachability calculation, attack graph modelling, core building, and analysis [12].

Reachability calculation examines network reachability, which determines if two given hosts can communicate with each other [13]. Attack graph modelling is the construction of attack templates that describe the elements of several attacks, vulnerabilities, and correlations between the vulnerabilities. Core building is the main algorithm used to generate an attack graph and aims to prune several paths. Finally, attack graph analysis is significant as it identifies the path that an attacker will most likely use to reach the target. Although researchers usually combine this stage with the core building stage, they should be conducted separately as they both use different methods and techniques and have different purposes. The main difference between the attack graph analysis and the core building stages is that the core building aims to prune the number of paths to reduce the complexity and time required to generate an attack graph, while the analysis stage aims to discover the optimal path that attackers may use to reach the target.

However, existing attack graphs are plagued with a few issues, one of which is scalability [14]. This is because an increase in network size increases the complexity of the network. Secondly, attack nodes need to be assigned manually. Therefore, this present study used an A* prune algorithm as well as the personal agent used in the attack graph generation to reduce complexity and generation time. The RF algorithm was then used to detect, predict, and dynamically locate attack locations to analyse the attack graph. The improvement will remarkably contribute to network security in handling issues of attack detection and prediction. An improved attack graph will increase the accuracy rate of detection, predicting the next step of the attack. This research focuses on enhancing the attack graph representation and analysis to find the optimal path that might use by the attacker to reach the target node.

In summary, the contributions of the paper are as the following:

- The naïve approach pruning algorithm is one of the simplest algorithms that is used to prune the graph edge, not only the simplest but also the fastest and more accurate algorithm. While Personal agents refer to computer programs that learn about a user's preferences, interests, and behaviours in order to provide them with proactive, personalised support through a computer application. This contribution achieved two parts: (1) provide a fast response from the node using a personal agent, which leads to the fast calculation of the reachability, thus will reduce the generation running time. (2) prune unnecessary edges from the attack graph to minimise the attack graph's complexity.
- Machine learning algorithms are commonly employed to categorise network traffic in an attempt to detect attacks. Four ideas are presented in this contribution: (1) Detect and predict a network attack, (2) dynamically discover the attack location in the network, (3) find the target node in the network based on the path steps between the attack node and the servers, (4) determine all attack paths between the attack node and the target node

The rest of this study is organised as follows: Section 1 discusses previous studies, while section 2 presents the attack graph model. Section 3 explains the proposed model, while section 4 describes the experiments. The results are explained in section 5, while section 6 provides a conclusion.

2. Related work

Recently, many attack graphs have been generated using different methods and techniques to enhance attack graphs. In this section, the most recent attack graphs will be discussed in two parts: attack graph generation and analysis that encompasses the previous stages.

2.1. Attack graph generation

In recent years, many attack graphs that use different methods and techniques have been proposed. In order to ensure that all computing agents are parallelising computing the load balancing, Li et al. [15] proposed a search forward attack graph using hypergraph partitioning. Chen et al. [16] proposed a supervised Kohonen neural network attack graph that could forecast attack status and success rate should the attacker successfully exploit the vulnerabilities. Yichao et al. [17] proposed a compact graph planning that depends on the attack path discovery algorithm that might use permeation testing goal information to prune unnecessary paths depending on another structure proposed by Frances and Geffner [18]. Guan et al. [19] used parallel distributed computing to construct sub-graphs and then combine them into the entire network attack graph. To compute the attack probabilities of every node in-network, vulnerability values, attacker information, attacker willingness, and attack skills to build the security evaluation index system were used. Ibrahim et al. [20] proposed a hybrid attack graph (HAG) that captures both the logical changes under attack as well as the real values of resilience parameters associated with the attacks that make up the graph. Shuo et al. [21] used heuristic searching in attack graph generation. A predicate logic was used to explain the attack patterns and network environment. A matching index table and attack graph structure were then used to generate the attack graph. Hong et al. [22] proposed using a multi-layer hierarchical attack representation model (HARM) to reduce computational complexity by modelling different components in different layers of the system. Li et al. [23] proposed that attack graph generation depends on state reduction while Cook et al. [24] generated large-scale attack graphs. In this present study, a serial algorithm was first used, followed by complexity analysis to identify bottleneck and parallelisation opportunities. A parallel algorithm was then used to compare the performance characteristics of these algorithms. Zhang et al. [25] established an attack graph-based quantitative assessment approach for ICS security. The data from a high-performance graph database is used to create a flexible attack graph model. The authors offered a complete calculating approach with a predefined selection strategy to locate critical paths in ICSs, whether in the form of a nested path or a parallel path, by leveraging graph indications on specific nodes and edges to obtain security metrics. Sabur et al. [14] present a scalable security state (S3) framework for the network based on segmentation. The framework divides the large scale network region into smaller, controllable portions using the well-known divide-and-conquer strategy. To partition the system into segments based on the similarity between the services, this work use a well-known segmentation approach derived from the K-means clustering algorithm. The segments are separated by a distributed firewall (DFW), which prevents the attacker from moving laterally and compromising them.

In summary, although different methods have been used to improve attack graph generation, attack graphs still suffer from a few issues, especially scalability.

2.2. *Attack graph analysis*

Hughes and Sheyner [26] were the first to use attack graphs to forecast cyber-attacks. Since then, several researchers have used the forecasts of attack graphs analysis to concentrate on traversing the graph and looking for an optimal attack path or on the probability of the graph's edges [27].

Wang et al. [28] recommended using the breadth-first search (BFS) on the attack graph, starting with the most recent warnings. The hunt proceeds from the latest warnings to find paths that meet all security requirements and exploits without considering the conjunctive and disjunctive correlations between exploits. The technique basically searches the attack graph for all the next attacks. Although the prediction results were not discussed, computational and memory use was addressed. In 2013, two alert correlation models, both including prediction, were proposed. Chung et al. [29] proposed NICE, a framework of countermeasure choices in virtual networks that employ threat graphs to project and model attacks. On the other hand, Kotenko and Chechulin [30] recommend CAMIAC, a framework for cyberattacks simulation and effect evaluation using attack graphs. Both methods use attack projection as part of a broader framework that this present study is not based on.

Fayyad and Meinel [31] described a prediction model using object-oriented data obtained from an attack graph that was constructed after network recognition. Data derived from various databases, namely, the integrated data store (IDS) database, the national vulnerabilities database (NVD), and data for the attack graphs, were used. Similar to the prediction model of this present study, the prediction mechanism predicted the attacker's next step and the attack scenario. Nevertheless, the weights of each state are manually allocated and are dependent on vulnerability exploitations.

Ramaki et al. [32] proposed a method for real-time warning correlation and prediction. The framework operated in two modes; online and offline. A

Bayesian attack graph (BAG) was built using low-level warnings in the offline mode. The intruder's next move in the online mode was then predicted using the offline BAG. The efficacy of the method was tested using the DARPA 2000 dataset. The prediction precision was found to increase over the duration of an attack case. The precision ranged between 92.3 per cent at the first attack stage to 99.2 per cent at the fifth attack stage.

Orojloo and Abdollahi Azgomi [33] combined attack graphs and fuzzy logic to predict attacker attitudes using a series of potential malefactor moves. The study described attack behaviours upon gaining access to the network and described the capability of attacks perpetrated by experts.

Polatidis et al. [34] combined attack graph analysis with technological recommendations to predict the next steps of an attack. The network was studied, and a graph of possible attack routes was created. A multi-level collaborative filter was employed to predict how the attackers may proceed after gaining access to some of the properties. However, the starting node still had to be assigned manually, and the proposed model could only predict the next steps of an attack once the network had been breached.

In summary, the studies examined in this literature review used attack graph models to predict attacks. More specifically, they were used only to predict the next step of an attack. These studies also needed to manually identify the starting node of an attack location in the network. The following section provides an overview of attack graph models to present a fuller picture of attack graph generation.

3. Proposed model

This section discusses the method of attack graph generation. As seen in Fig. 1, the generation process is divided into four phases. The first step determines the reachability of an attack graph. This entails determining reachability conditions between hosts. The modelling phase explains the extent of an attack graph's configuration and how individual attack models could be built. The attack graph is then generated by calculating the potential attack paths during the core building phase. At this stage, several useless paths are pruned. The analysis is then used to determine all the possible paths of an attack, from the starting to target nodes. Every phase will be discussed in greater detail in the subsequent section.

3.1. Reachability calculation

Reachability calculation uses the information in the network to find the paths between hosts. As such, the filtering devices rule of the network must be extracted and modelled. Reachability calculation can be performed in many ways. This present study used a multigraph and connection matrix to calculate the reachability of the nodes in the network. The purpose of calculating reachability using a connection matrix and multigraph was to reach each IP address and the port of each node in the network.

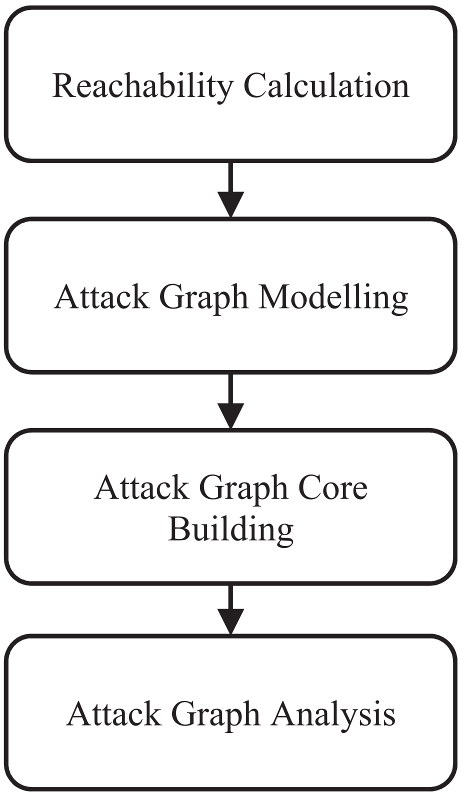

Fig. 1. Attack graph generation phases [35].

A connection matrix contains the reachability of each node in a network. The row represents the source, while the column represents the target. If there is a connection between the nodes, the value will be 1. It is 0 if there is no connection between the nodes. Each node calculates its reachability and sends this data to the administrator via a personal agent.

Multigraphs allow for multiple edges (or parallel edges) [36]. This implies that edges have the same end nodes. However, two vertices can be connected by more than one edge. A multigraph is distinct from a hypergraph as it connects any number of nodes, not just two.

In this present study, the reachability of the multigraph depicted the connectivity conditions between the software applications installed in the hosts. The firewall rules, access control, trust relations and security policies of the software applications were used to define the reachability information.

These factors, as well as other information, were collected from the nodes via a personal agent and sent to the administrator server, which is responsible for constructing the attack graph. Every personal agent was responsible for collecting the data required to calculate reachability. Personal agents were also responsible for updates to reduce time and complexity. The administrator server was responsible for defining the attacker's privileges and the vulnerability exploits of several software applications.

The graph-vertex indicates the nodes in the network. On the other hand, a graph edge depicted source and destination software applications that could connect directly with each other if certain requirements were provided. These conditions permitted direct accessibility between the software applications, which were stored on every edge.

3.1.1. Personal agent

A personal agent is a computer program that learns the users' preferences, interests, and habits to provide them with proactive assistance through an application [37]. This present study employed a personal agent between the nodes and the administrator node. The job of a personal agent is to save the reachability and the information of each node and send it to the administrator to complete the reachability calculation. Apart from that, if any update occurs in the node and it cannot connect to the server to send new information, the personal agent keeps using the old information until the connection with the server is restored, at which stage it will send the new information. Fig. 2 shows the flowchart of a personal agent.

3.2. Attack graph modelling

Modelling attack templates and determining the structure of attack graphs are all aspects of attack graph modelling. The pre-and post-conditions for the vulnerabilities are described in the attack template modelling. It also provides a mechanism for calculating these criteria for individual vulnerabilities using data from publicly available vulnerability and weakness databases. Choosing nodes and edges types in an attack graph is part of evaluating the attack graph structure.

3.2.1. Attack template modelling

Usually, the attack template is derived from the vulnerabilities data provided by CVE and NVD, and the formalization method expresses the attack

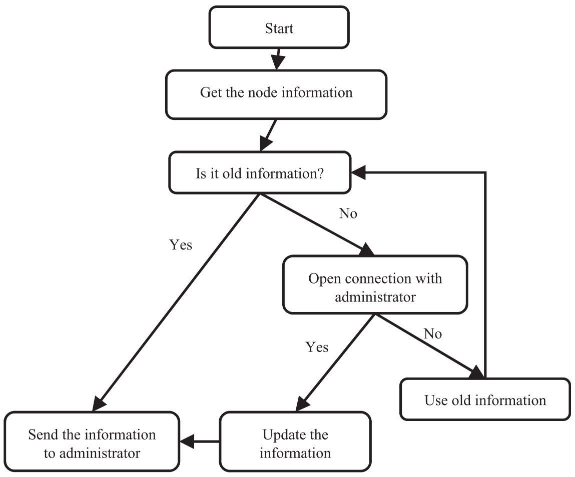

Fig. 2. The personal agent process flowchart.

template factors. The attack template was used for the first time by Jha et al. [38], which suggested and explained the attack template and divided it into four categories: attacker pre-conditions, network pre-condition, attacker post-condition, and network post-condition. In this paper, we use the same attack template in Refs. [15,39]. Fig. 3 shows the attack template that used in our work.

Definition 1. Condition refers to the information that could be gained using software applications. It has two elements which are *Category* represent the gains on the application, *Host* refer to the location of the software applications.

Definition 2. Direct condition assigns an additional element to the condition. This contains *CPEId*, which represents the CPE identifier of the software.

Definition 3. An indirect condition assigns an additional element to the condition. This contains *ProductType*, which represents the product type of the software.

Definition 4. Vulnerability in this work is defined as a single CVE entry defined in the CVE database.

3.2.2. Attack graph structure

The structure defines the nodes and edges of the attack graph generation. We will use the attack graph structure proposed by Kaynar and Sivrikaya [39], as shown in Fig. 4.

Definition 5. Privilege node refers to the attacker privilege on software applications on a network host.

Definition 6. A vulnerability conjunction node denotes a conjunction connector for more than one vulnerabilities node in the attack graph.

Definition 7. A vulnerability node refers to a vulnerability in a software application on a network host.

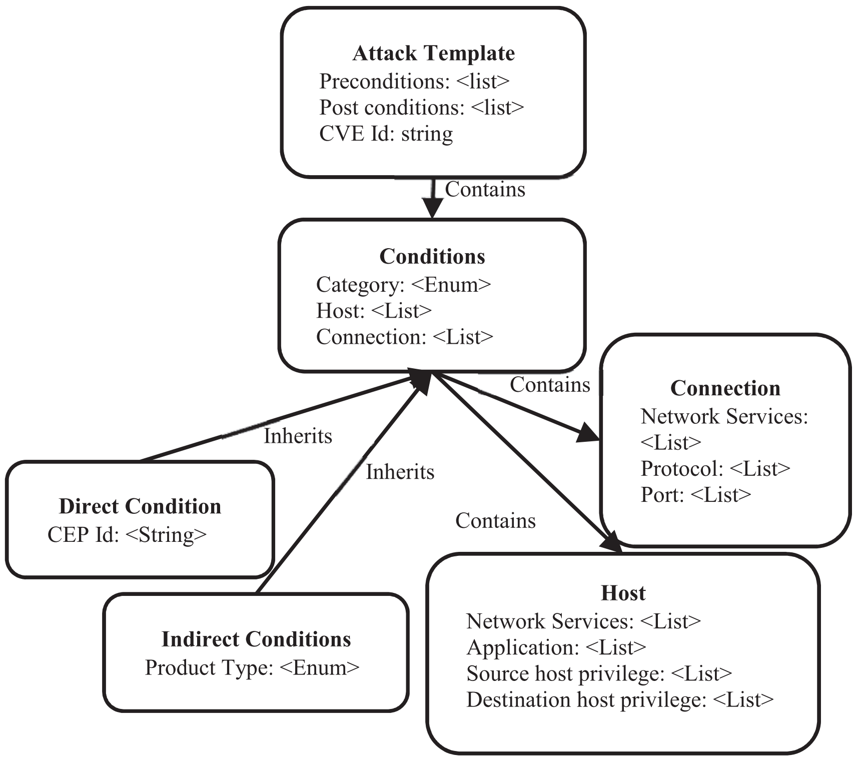

Fig. 3. Attack template [15,39].

3.3. Attack graph core building

Core building refers to the main algorithm process of the attack graph that was developed in this present study. It is viewed as a search problem that is solved using a personal agent. A reachability multigraph was first created for this purpose. The A* pruning algorithm was then implemented on the reachability information contained in the multigraph. The core building process can be divided into two stages; attack graph generation and pruning. These two stages are explained in greater detail in the subsequent section.

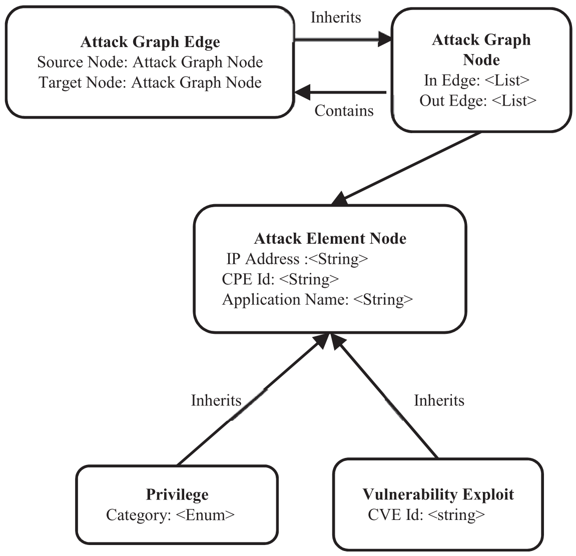

Fig. 4. Attack graph structure [39].

3.3.1. Attack graph generation

An administrator generated an attack graph after the data of each node provided by the personal agents was collected (Fig. 5). The algorithm began by extracting the information of every node in the network. This information included reachability, vulnerabilities, privileges, application list, and connection information. The mutual vulnerability between the nodes and the nodes in the reachability list of the node was then checked. If more than one vulnerability connected the node and the reachability nodes list, the node was labelled a conjunction node. If only one vulnerability connected the node and the reachability nodes list, it was labelled a vulnerability node. Lastly, if no vulnerabilities were connected to the node, it was labelled a privileged node.

The attack graph was then updated prior to the implementation of the pruning algorithm. This update is of utmost importance during attack graph generation as the personal agent may receive some

```
1. Inputs:
node_list; // network nodes list
Vul_list; // vulnerabilities list;
P=0;
2- Process
Foreach node in node_list
    M = getinformation(node);
    M.P = 0;
    Foreach n in M.reachability
        F = getinformation(n);
        If (check_vulnerabilities(M.vulnerabilities, n.vulnerabilities) ≠ 0)
            M.P++;
            Foreach vulnerability in check_vulnerabilities (M.vulnerabilities,
            n.vulnerabilities)
                Graph.addedge(node, vulnerability);
                Graph.addedge(vulnerability, n);
            Endfor
        Else
            Graph.addedge(node, requireprivilege);
            Graph.addedge(requireprivilege, n);
        Endif
    Endfor
    If (M.P == 0)
        Privilege_nodes.add(node);
    Else if (M.P == 1)
        Vulnerability_nodes.add(node);
    Else if (M.P >1)
        Conjunction_nodes.add(node);
    Endif
Endfor
3. Output:
Attack graph, Vulnerability_nodes, Privilege_nodes, Conjunction_nodes
```

Fig. 5. Attack graph generation.

privileges or vulnerabilities only after the generation is complete. If there are any changes in the information of any of the nodes, for instance, gaining privileges or discovering a new vulnerability caused by installing an application, the personal agent sends this information to the administrator to dynamically update the attack graph (Fig. 6).

3.3.2. A* pruning paths

A*-search is a well-known searching strategy in Artificial Intelligence. This present study used the A* prune as presented by Liu and Ramakrishnan [40], with some changes to the definitions and steps. A* prune combines A* search algorithm and proper pruning techniques to better prune algorithms. The definitions of the algorithm are as follows:

Definition 8: Consider a network depicted by graph $G=(V,E)$, where V refers to a set of vertexes and E refers to a set of edges. Every edge $(i,j)\in E$ is related to R, not negative and QoS values: $w_r(i,j)$, $r=1,2,\ldots,R$. A length function w_0 was determined as follows:

$$w_0(i,j)=\sum_{r=1}^{R}a_r w_r(i,j) \tag{1}$$

Definition 9: Additive Parameters: If the $W_r(p(i,j))$ the parameter is associated with the $p(i,j)$ path, and $W_r(p(i,j))=w_r(i,j)$ when $p(i,j)$ is one path, then w_r is an additive parameter if

$$W_r\left(p(i,\,u)\,p(u,\,j)\right)=W_r\left(p(i,\,u)\right)+W_r\left(p(u,\,j)\right) \tag{2}$$

Definition 10: Tail path and head path: Supposing that node u is an intermediate node of the path $p(i,j)$. Node u divides the $p(i,j)$ path into two; path $p(i,u)$ and path $p(u,j)$. The $p(i,u)$ path is known as the head path while the $p(u,j)$ path is the tail of path $p(i,j)$. Therefore, the $p(i,j)$ path is expressed as a combination of the tail path and head path given in the format below:

$$p(i,j)=p(i,u)p(u,j) \tag{3}$$

Definition 11: App edge: If app m defines the edge between nodes i and j, m is an application of a set of the joint application list M between nodes i and j

$$m\in M, M(i,j)=\{app_1,app_2,\ldots..,app_n\} \tag{4}$$

Beginning with path $p(s,s)$, in which s denotes the source node, although the A*Prune algorithm can theoretically reach all the paths in $P(s,V)$, only paths in the admissible head path set $P(s,V,H(p),C)$ remain as candidates for future expansion after sufficient pruning using the constraints C. Moreover, the candidate paths are organised in such a way that the path with the shortest projected length $H_0(p)$ are chosen and expanded first. The expansion is halted once a sufficient number of constrained-shortest-paths (CSPs) are identified or there are no more candidate paths. Only a portion of the admissible head path (AHP) set $P(s,V,H(p),C)$ are

```
Function updateattackgraph (attack graph, vulnerabilities_nodes,
conjunction_nodes, privilege_nodes, node_id)
{
    M = getoldinformation(node_id);
    H = getnewinformation(node_id);
    If (check_vulnerabilities (M.vulnerabilities, H.vulnerabilities) != 0)
        Foreach node in H.reachability
            Foreach (vulnerability in check_vulnerabilities (H.vulnerabilities,
            node.vulnerabilities))
                Graph.addedge(node_id, vulnerability);
                Graph.addedge(vulnerability, node);
                H.P= M.P+1;
            Endfor
        Endfor
    Endif
    If (H.P == 1)
        Vulnerability_nodes.add(node_id);
        Privilege_nodes.remove(node_id);
    Else if (H.P >1)
        Conjunction_nodes.add(node);
        Vulnerability_nodes.remove(node_id);
    Endif
}
```

Fig. 6. Attack graph update.

expanded using the proposed approach. Fig. 7 depicts the A*Prune algorithm pseudocode.

The first step of the algorithm is to obtain the full attack graph with V; the nodes; and E; the edges; as well as the list of networks (*vul_list*), the application list (*app_list*) for all nodes, and the number of paths in the full attack graph.

Assuming that *AHP_heap* has been obtained, an acceptable head path list is initialized with the trivial path $p(s, s)$. The path expanding procedure then picks and eliminates path p from *AHP_heap*, then expands the chosen path to obtain all the available extended paths and inserts all allowable head paths into *AHP_heap*.

Once the process chooses the first path in *AHP_heap*, it is removed from the *AHP_heap* before each node in the path is checked by the vulnerabilities and application lists of each node. If there is compatibility, an edge is added between these nodes until the target t is reached. This process is repeated for all paths in the attack graph until a *CSP_list* is returned, which contains all paths in the attack graph after it has been pruned.

```
1. inputs:
$G = (V,E)$;
$(s,t)$: a node pair;
$K$: number of paths;
$applist$: List of application;
$vul\_list$: vulnerabilities list;
$R$: number of constraints;
$k \leftarrow 0$ ; // number of CSPs found
$AHP\ heap \leftarrow \{p(s,s)\}$;
$CSP\ list \leftarrow \{\}$;
2. Process
while ($k \le K$ and $AHP\_heap$ is not empty)
    $q(s,u) \leftarrow$ the first path of $AHP\_heap$;
    Remove the first path of $AHP\_heap$;
    $u \leftarrow$ the end node of $q(s,u)$;
    $sourceapp\_list[] \leftarrow applist[s]$ ;// source application list
    $distapp\_list[] \leftarrow applist[u]$;
    $vul\_list = findexploitvulnerabilities\ (sourceapp\_list, distapp\_list)$;
    $sourcevul\_list\ [] \leftarrow vul\_list[s]$;
    $distvul\_list[] \leftarrow vul\_list[u]$;
    $gainprivileges = findgainprivileges(s,u)$;
    if $gainprivileges$ !=0 then
        $addedge\ (s, gainprivilege)$;
        $addedge\ (gainprivilege, u)$;
    else if $vul\_list! = 0$ then
        $addedge\ (s, vul\_list)$;
        $addedge\ (vul\_list, u)$;
    else
        go to 2;
    end if
    if $u = t$ then
        insert $q(s,u)$ into $CSP\_list$;
        $k \leftarrow k + 1$;
        go to 2;
    else
        go to 2;
    end if
end while
3. Output
return $CSP\_list$
```

Fig. 7. A* prune algorithm.

3.4. Attack graph analysis

The attack graph analysis will be discussed in this section. The attack prediction and attack projection for the path from the attack location until the target will be explained in detail. Fig. 8 shows the attack graph process. Each step will be discussed in detail below.

3.4.1. Attack prediction

In this section, the attack prediction using machine learning will be explained. The aim of using attack prediction in the attack graph is to find the attack location dynamically. Fig. 9 shows the attack prediction process using machine learning.

3.4.1.1. Data collection and preparation. In the attack prediction process, network traffics will be used. There are many datasets used in detection and prediction like the 913 MALICIOUS dataset [41], CSE-CIC-IDS2018 and KDD, also there are many techniques are used to generate the traffics like Ostinato, Genesids and Cisco TRex [42]; however, the data used in this work is CIC-IDS2017 because it is large and has many types of attacks. The CICIDS-2017 was established by the Canadian Institute for Cybersecurity at the University of New Brunswick. Note that it has 85 attributes. The data collection period took five days, beginning from Monday

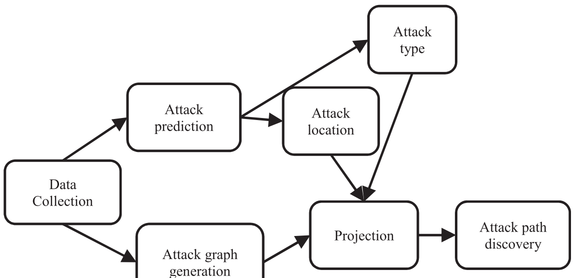

Fig. 8. Attack graph analysis.

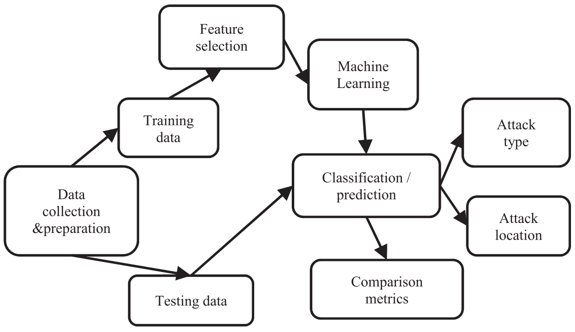

Fig. 9. Attack prediction process.

(2017, July 3) at 9 a.m. and ended at 5 p.m. Friday (2017, July 7). Attacks include DoS, Botnet, Web Attack, Brute Force FTP, Brute Infiltration, DDoS and Force SSH. More details about the dataset can be found in Ref. [43].

3.4.1.2. Feature selection. A research question in network intrusion detection questions how to choose appropriate features. Discovering critical attributes not only speeds up data manipulation but also has the potential to increase identification rates. There are several attributes in selection algorithms.

The most widely used feature selection method is Information Gain [44], which is a feature selection dependent on filters [45]. Information Gain eliminates noise generated by irrelevant features by employing a simple attribute rank. It then determines a feature with the most information base in a particular class. Calculating the entropy of a feature determines which one is the greatest. Bereziński et al. [46] defined entropy is a measure of uncertainty that may be employed to infer the features' distribution in a succinct manner. The entropy may be determined using the following Eq. (5):

$$Entropy(S) = \sum_{i}^{c} -P_i \, log_2 \, P_i \tag{5}$$

where c denotes the number of values in the classification class while P_i denotes the number of samples for class i. The Information Gain value is determined after the entropy value has been obtained by employing Eq. (6):

$$Gain(S,A) = Entropy(s) - \sum_{Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{6}$$

where S denotes a sample, A denotes an attribute, v represents a potential value for attribute A, while $Values(A)$ denotes a set of potential values for A. Moreover, $|S_v|$ refers to the number of samples for value v, $|S_v|$ is the number of samples for all data samples, while Entropy (S_v) is the entropy for the sample with a value v.

3.4.1.3. Machine learning algorithm. Machine learning algorithms have been widely used for several classification and prediction problems and have provided accurate results. In this work, the Random Forest algorithm is used to classify the network traffic to detect, predict and find the attack location, as in Fig. 10.

```
1. Input:
A training set $S := (x_1, y_1); \dots ; (x_n, y_n)$, features $F$, and number of trees in forest $B$
2. Process:
Function RandomForest (S, F)
    $H \leftarrow \theta$
    For $i \in 1, \dots, B$ do
        $S^{(i)} \leftarrow$ A bootstrap sample from $S$
        $h_i \leftarrow$ RandomizedTreeLearn($S^{(i)}$, $F$)
        $H \leftarrow H \cup \{h_i\}$
    End for
    Return $H$
End Function
Function RandomizedTreeLearn (S, F)
    At each node:
        $f \leftarrow$ very small subset of $F$
        Split on best feature in $f$
    Return The learning tree
End function
```

Fig. 10. The Random Forest training algorithm.

One of the ensemble classifier approaches is Random Forest (RF). If a decision tree classifier is used in an ensemble, the classifiers are called a forest. Each decision tree is generated by selecting attributes at random at each node for separation [47]. Breich suggested the RF algorithm in 2001 [48]. It has been used in some anomaly detection experiments, for instance, research performed by Lallie et al. [49].

3.4.1.4. Attack location. The location of the attack in the network is important to identify the starting node for the attack to find the path that the attacker might use. In this matter, the location will be identified using machine learning algorithms as in Fig. 11.

After the RF is trained, the administrator will monitor the network and capture the traffic and send it to RF to test it. The algorithm will split the network traffic depending on the IP addresses and then test the traffic based on the IP address. If there is an attack in the traffic, the algorithms will return the address and attack type. After identifying the location of the attack, the algorithm will find the

```
1. Input:
    Network Traffics
2. Process:
Classify the network traffics based on IP address
Foreach IP address Do
    Test the traffic using ML
    if (traffic! normal)
        X= IP address
        Y= attack type
    Endif
Endfor
i =0;
Foreach data servers do
    Array[i] = min_steps (X, data server)
    i++
Endfor
target = min(array)
3. Output
        Return X, Y, target
```

Fig. 11. Attack location pseudocode.

nearest data server to the location to identify it as a target node for the attacker.

To find the location of the attack, the RF will construct the node's location based on the coming traffic. Once the algorithm assigns that there is an attack on the traffic, the algorithm will analyses the traffic information to construct the IP address of the attack node and the type of the attack. After the traffic has been classified, the system will get the IP address of the node that has been attacked or is still under attack; then, the system will apply rules to block this node from the network. Fig. 12 shows the algorithm that has been used to block the node in the network.

Once the system confirms there is an external attack on the network and applies the rules to block the node that got attacked, the system will load the node information from the attack graph to find the capability of the attack.

3.4.1.5. Performance measure. To calculate the performance of RF, first of all, the algorithm will classify the dataset into four types which are True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). Using these four factors, Precision and recall for the algorithm will be calculated. Precision is the ratio of the attack flows (TP) to the characteristic flows (TP + PF), as in Eq. (7):

$$Precision = TP/(TP + FP) \tag{7}$$

Meanwhile, Recall or Sensitivity is a ratio of correctly identified attacks (TP) with the overall predicted flows (TP + FN) as in Eq. (8):

$$Recall = TP/(TP + FN) \tag{8}$$

If the model has been established, it could be employed to recognize possible targets for traffic attacks. We utilized our developed model to forecast attacks based on traffic classification during the testing process. If the attacker did indeed target a host as expected by the machine learning algorithm, the model is deemed correct. Eq. (5) is employed to measure the model's accuracy, given by:

```
Input: Attack location
X= attack location
If (predict the attack on X)
Create new rule in firewall;
rule. direction = net_fw_rule_direction_.net_fw_rule_dir_in;
rule. RemoteAddress = "X";
rule. Grouping = "@firewallapi.dll,-23255";
rule. Profiles = fwPolicy2.CurrentProfileTypes;
rule. Action = net_fw_action_.net_fw_action_block;
fwPolicy2.Rules.Add(rule);
Y= "system under attack";
Return (Y)
Else
Y= "system is safe";
Return (Y)
Endif
```

Fig. 12. Blocking attack node algorithm.

$$Accuracy = \frac{Number\ of\ predict\ attack}{number\ of\ the\ attack} \times 100 \tag{9}$$

To test the RF performance, in this paper, the CICIDS-2017 dataset will be used. The CICIDS dataset contains eight files, each file collected at a different time and has different attacks. The dataset includes 78 function columns and a one-mark column, which were included in this study. The dataset includes two "Fwd Header Length" features or columns, all of which are redundant, so one must be omitted. After deleting the obsolete features, there are only 77 features left to evaluate [44].

3.4.2. Attack projection

Attack projection defines the attack's next step inside the network. It is used to project the resumption of an attack and predict upcoming events. To predict the attack's next step, first, we use the attack location in the network, besides the vulnerabilities list, privileges list and network connection. The rule-based methods will be used to predict the next attack step.

Attackers can get unauthorized access to the system by using basic privileges that satisfy some initial input requirements. In general, attackers can gain access to information systems by exploiting several vulnerabilities, like software vulnerabilities.

The attack path discovery pseudocode is shown in Fig. 13, while the following activities need to be executed for the algorithm to determine the attack paths:

1. Entry Points: the entry point will be the attack location. After we find the location of the attack in the network, it will be the point to start to find the all paths from this point until reaching the target.
2. Vulnerability Chains (VC): In VC, this work uses a rule-based reasoning approach to generate a

```
1- Input:
 attack graph, attacker location, Vulnerabilities list, Privileges list, Network
information, target node
X= Attacklocation();
K= getinformation(X);
Vul_score =0;
scored=0;
2- Process
$Constructgraph(X, target)$;
Foreach (nodes in K.reachability)
If(gainprivileges(x,node) || checkvulnerabilities (x,node) !=0)
scored = score(x, node);
Endif
If ( scored > vul_score)
Next_step = node_id;
Vul_score= scored;
Endif
Endfor
3- Output
Return Next_step
```

Fig. 13. Attack path discovery.

```
Constructiongraph (s, d)        // s = attack location, d = attack target
{ bool* visited = new bool[N];              // Mark all the nodes as not visited
int* path = new int[V];                  // Create an array to store paths
int index = 0;                          // Initialize path[] as empty
for (int i = 0; i < V; i++)             // Initialize all vertices as not visited
visited[i] = false;
printAllPaths (s, d, visited, path, index); }      // call to print all paths
printAllPaths (int u, int d, bool visited[], int path[], int& index)
{ visited[u] = true;
path[index] = u;
index++;
if (u == d)    // If current vertex is same as destination, then print current path
{ for (int i = 0; i < path_index; i++)
cout << path[i] << " ";
cout << endl; }
else                              // If current vertex is not destination
{ for (i = adj[u].begin(); i != adj[u].end(); ++i) // Recur for all the vertices
adjacent to current vertex
if (!visited[*i])
printAllPathsUtil(*i, d, visited, path, index); }
// Remove current vertex from path[] and mark it as unvisited
index--;
visited[u] = false;
}
```

Fig. 14. Construct attack graph between two nodes [50].

chain of vulnerabilities on various assets that result from multi-step attacks launched from the source to exploit the target vulnerabilities.

To find the attack's next step and all possible paths, first, the algorithm starts to get the attack location, attack target, attack graph, vulnerabilities list and privilege list as an input. In the beginning, the algorithm starts to generate an attack graph from the attack location to the target nodes by using the construction function as explained in Fig. 14.

Then the algorithm tries to find the possible attack's next step by using the vulnerabilities list and privilege list. For all nodes connected to the attack's location node, check the vulnerabilities and gain privileges; if there are gain privileges and vulnerabilities that might be exploited, then calculate the score of the gain privilege and vulnerabilities.

After finding the higher score between all the nodes, the algorithm will return a recommendation to the administrator with the attack paths and the possible next step for the attack, as shown in Fig. 15.

```
Score (node, attacklocation)
{
S = 0;
If ( gainprivilege(node, attacklocation)!= 0)  // if there is gain privilege,
add 10
If ( checkvulnerabilities(node, attacklocation) !=0)
{
S = 10 + vulnerabilitiesscore(node, attacklocation);
}
endif
Else if ( checkvulnerabilities(node, attacklocation) ==0)
{
S = 10;
}
Return (S);
Else If ( gainprivilege(node, attacklocation)== 0)
If ( checkvulnerabilities(node, attacklocation) !=0)
{
S = vulnerabilitiesscore(node, attacklocation);
}
endif
Else if ( checkvulnerabilities(node, attacklocation) ==0)
{
S = 0;
}
Return (S);
}
```

Fig. 15. Vulnerabilities score calculation.

4. Experiment

The experiment was conducted on a typical enterprise network to test our proposed attack graph. The generated attack graph was tested using the network topology portrayed in Fig. 16. The network contains three servers which are a webserver, file server and database server; it also has an IDS device and four workstations, two workstations are working with windows 10, and the other two workstations are using windows 7. There is a perimeter firewall. The following firewall rules regulate network connectivity in this network topology:

- The network includes bidirectional connectivity between the workstations and the webserver.

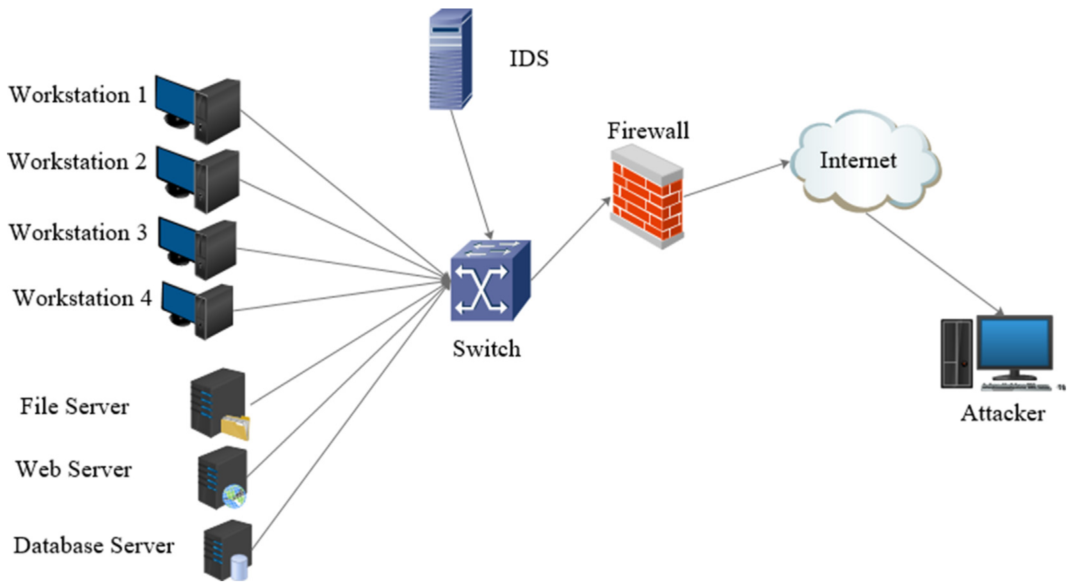

Fig. 16. Experiment network topology.

- There is bidirectional connectivity between workstation 1 with workstations 2, 3 and 4.
- There is bidirectional connectivity between workstations 3 and 4.
- The attacker's host is connected to the Internet and has HTTP protocol and HTTP port access to the webserver.
- The four workstations and fileserver have access to each other through the NFS protocol and NFS port.
- The HTTP protocol and HTTP port provide internet access to the four workstations and file servers.
- Workstation 4 has access to the network database server.

The experiments used to evaluate the generating attack graph in this environment uses the CPU with core i5 2.0 GHz with 8 GB of RAM. The operating system is Windows 10, while the coding was done using Microsoft Visual Studio C# 2012. The attack graph generation for the network topology above is shown in Fig. 17.

5. Results

The findings of the experiment are explained in the subsequent section in two parts; attack graph generation and analysis.

5.1. Attack graph generation

The results indicate that a personal agent is able to collect data and send it to the administrator as well as reduce latency, leading to no missing data when the attack graph is updated, if any information has been changed. The A* prune algorithm reduced the generation time and complexity of an attack graph. As seen in Fig. 16, the running time for network topology after using the A* prune algorithm was 0.3 s.

These results encouraged the testing of more factors in the attack graph as well as larger numbers of nodes and servers to test the running time and complexity of the attack graph. Table 1 shows the results of each stage of the attack graph generation.

In Table 1, the running time of different stages of attack graph generation is explained. We can see the advantage of using the A* pruning algorithm and personal agent in decreasing the generation time, especially the personal agent, which reduces the reachability calculation in different network sizes.

Other pruning algorithms were used to generate the attack graph; however, it was determined that the A* prune algorithm provided the best and most accurate results among the tested algorithms (Fig. 18). The performance of the A* prune algorithm used in this present study was compared with that of depth-first search, breadth-first search, and greedy algorithm on the same number of nodes and topology. The results show that the A* prune algorithm provided better results than the other algorithms. The reason is that the A* pruning algorithm expands far fewer nodes than other algorithms. In this case, the paths will be found faster than other algorithms.

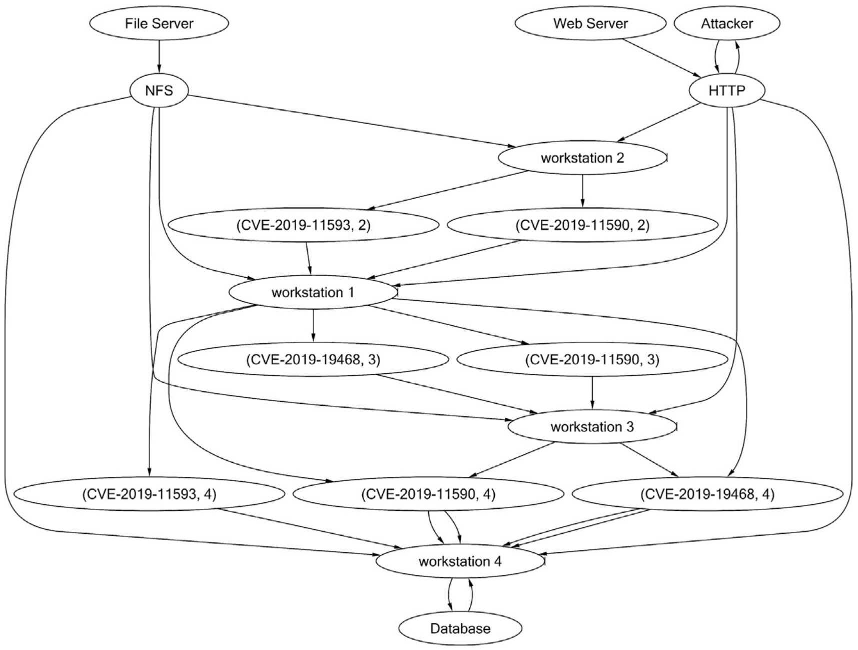

Fig. 17. Example of the attack graph.

Table 1. Running time of the attack graph (seconds).

Nodes Number	Full graph	Full graph with A*prune	Full graph with parallel	Our work
8	1.604	1.332	0.527	0.232
50	67.489	58.120	10.269	8.173
100	146.48	118.591	23.402	16.160
250	305.631	261.017	57.594	41.018
400	737.17	598.591	102.907	91.67
600	1278.816	1038.36	201.398	147.739
750	2103.479	1772.831	348.682	261.948
1000	4891.529	4167.192	603.481	479.018

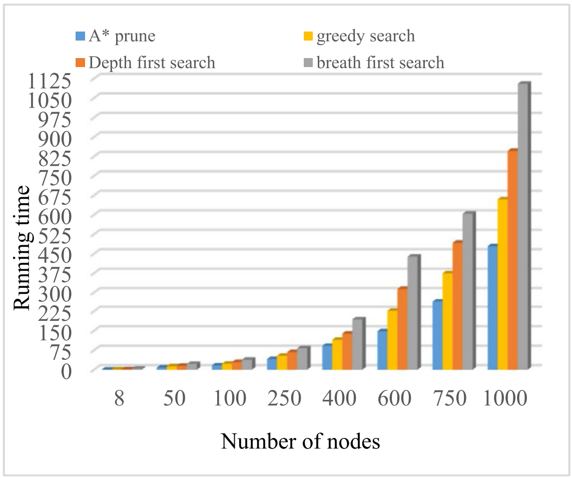

Fig. 18. Comparison results.

Four different network sizes were used to test the results obtained by extant studies with that of this present study, which used the A* prune algorithm. The results indicated that the model adopted by this present study provided faster generation times than that of extant studies. This was because this present study used the A* prune algorithm, which is faster than the other algorithms. This present study also used a personal agent to facilitate faster information reachability (Table 2).

Comparing the results, our work is faster than others because [39] depends on multi-agent to calculate and generate the subgraphs and then combine them in the administrator to generate the full graph. While Li et al. [51] depending on the size of the queue to represent the number of threats to decrease the generation time, however, merge the sub-graphs is another issue because it requires a long time to generate the full graph. While Feng et al. [52] depending on a decrease in the number of vulnerabilities in each node to ensure the attack graph will not be larger so it will lead to a decrease in the generation time, however, the attack graph is supposed to represent all the vulnerabilities in the network, so once the

Table 2. Comparison with previous work.

Network size	[39]	[51]	[52]	Our work
8 nodes	4.102	9.35	5.616	0.232
50 nodes	9.83	13.61	11.381	8.173
100 nodes	14.67	21.583	16.62	16.160
250 nodes	47.38	52.172	49.521	41.018
Complexity	$O(N^2/log(N))$	$O(\lvert Q_d\rvert N_e/n + k_1\lvert Q_d\rvert)$	$O(n*m^2)$	$O(n*m)$

vulnerabilities number increased, the generation time will be increased.

5.2. Attack graph analysis

The predictions and projections are explained in this section. In terms of attack prediction, the CICIDS2017 dataset was used to test the performance of the RF algorithm. Different criteria were used to test the performance of the RF algorithm, namely, the accuracy of detecting an attack, running time, recall, and precision. Table 3 compares the performance of the RF algorithm using the CICIDS2017 dataset with that of other machine learning algorithms, such as artificial neural network (ANN) and support vector machine (SVM).

The RF algorithm was found to provide higher average accuracy (98.1%) than the ANN algorithm (92.5%) and the SVM algorithm (74.7%). However, for the detection time, the SVM algorithm had the fastest running time (7.8 seconds), while the RF and ANN algorithms had running times of 8.9 seconds and 132.5 seconds, respectively, as shown in Fig. 19.

The RF consist of multiple single trees, each based on a random sample of the training data. So it is typically more accurate than other algorithms. Also, the RF is second faster to train because it is working only on a subset of features, so we can easily work with hundreds of features and a larger dataset. While SVM and ANN do not perform very well with a large dataset and when the dataset has more noise.

For attack projection, the network depicted in Fig. 16 was used. Fig. 17 shows the attack graph of the network, while Fig. 19 presents the attack path from the attack to the database server.

Table 3. Performance of machine learning algorithm in a 70:30 split dataset.

Detection	Random Forest	SVM	ANN
DoS/DDoS	99.9	90.6	99.4
Port Scan	100	98.5	99.9
Bot	98.8	77.6	99.9
Web Attack	99.9	66.9	84.2
Infiltration	90	50	75
Brute Force	100	65	96.6
Average	98.1	74.7	92.5
Running Time	8.9	7.8	132.5

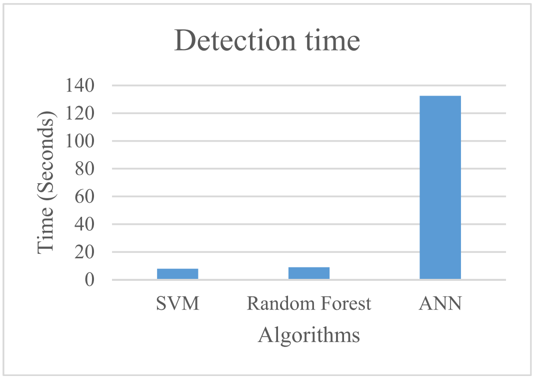

Fig. 19. Attack detection time.

If an attacker only has access to a webserver on this network, the attacker's first step is the webserver. From the webserver, there are only four ways for the attack to travel from the workstations to the target, i.e., the database server (Table 4).

Here, *A* denotes the attacker, *W* denotes the webserver, *D* denotes the database, and the numbers symbolise the workstations. Only the webserver can provide the attacker with access to the workstations. In this situation, the attacker's subsequent move is to attack the webserver. However, as the webserver has a bidirectional connection to the workstations, it cannot directly connect to the database server. This can only be accomplished via the workstations in a network. In this scenario, the attacker must first conquer the workstations before attempting to conquer the database server by exploiting network vulnerabilities.

The accuracy and running time of the machine learning algorithms are first measured, followed by system execution time. Fig. 20 displays the running time of the system, starting with the generation time for the attack graph, the detection time, and the

Table 4. Attack path.

Path number	Path
1	A-W-4-D
2	A-W-1-4-D
3	A-W-3-4-D
4	A-W-1-3-4-D
5	A-W-3-1-4-D

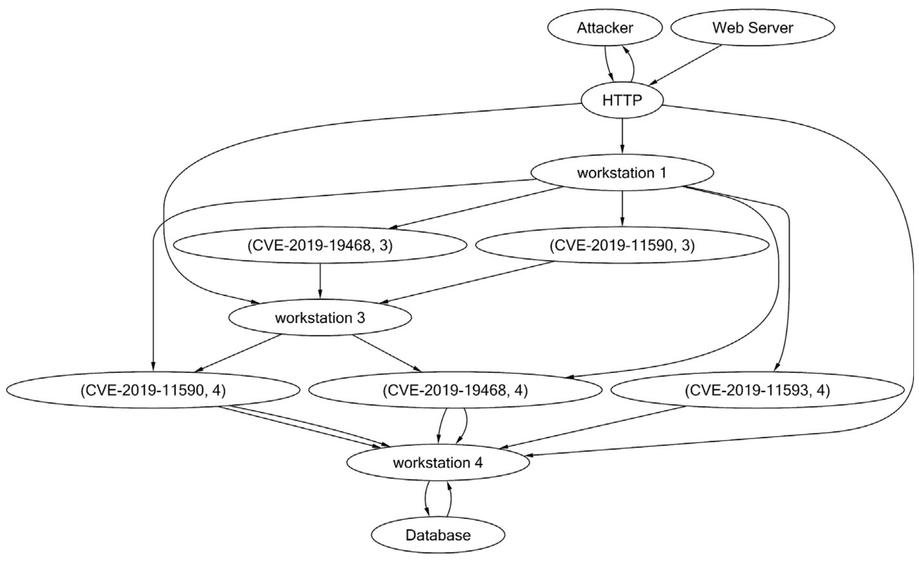

Fig. 20. Attack path discovery.

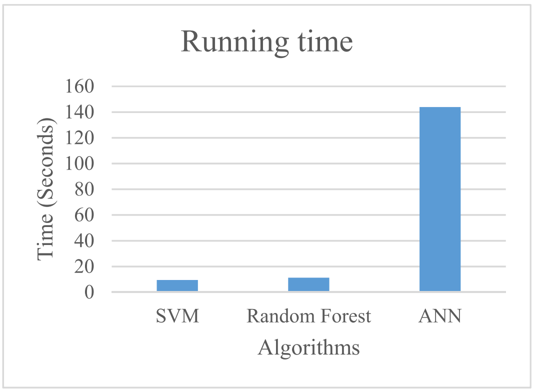

Fig. 21. System running time.

projection until the optimal path is found. The results indicate that the SVM algorithm system had the shortest running time (9.5 seconds), while the RF algorithm took 11.2 seconds and the ANN algorithm took 143.8 seconds (see Fig. 21).

6. Conclusion

There is an increased need for reliable attack identification as network threats become more complex and diverse. As cyber-attacks cannot be completely eliminated, the cybersecurity industry geared its research and development efforts towards detecting and minimizing the damage of security breaches. There is an increasing trend of developing more constructive security approaches that help deter or minimize security events before they cause more harm. As such, many approaches have been proposed in recent years. One of these approaches is the attack graph. In this present study, a personal agent and the A* pruning algorithm were used to improve attack graph generation by reducing the running time and latency of collecting and updating the data from the host. The RF algorithm was also used to improve attack graph analysis by detecting and predicting an attack and its location to project the next steps of an attack. The results showed that the attack graph developed in this study produced better results than the current attack graph. The results of the analysis indicated that the RF algorithm had higher accuracy than the ANN and SVM algorithms. Therefore, future studies could explore the use of different algorithms and techniques to reduce generation time as well as use more information to calculate reachability. Future studies may also use different attack prediction methods to increase the accuracy of finding the attack location in a network.

References

[1] A.A. Mutlag, M.K.A. Ghani, M.A. Mohammed, A. Lakhan, O. Mohd, K.H. Abdulkareem, B. Garcia-Zapirain, Multi-agent systems in fog−cloud computing for critical healthcare task management model (CHTM) used for ECG monitoring, Sensors. 21 (2021) 6923−6939, https://doi.org/10.3390/s21206923.

[2] Z.J. Al-Araji, S.S.S. Ahmad, M.W. Al-Salihi, H.A. Al-Lamy, M. Ahmed, W. Raad, N.M. Yunos, Network Traffic Classification for Attack Detection Using Big Data Tools : A Review, Intelligent and Interactive Computing, 67, Springer, 2019: pp. 355−363, https://doi.org/10.1007/978-981-13-6031-2_37.

[3] J. Jang-Jaccard, S. Nepal, A survey of emerging threats in cybersecurity, J Comput Syst Sci. 80 (2014) 973−993, https://doi.org/10.1016/j.jcss.2014.02.005.

[4] Y. Yang, L. Wu, G. Yin, L. Li, H. Zhao, A survey on security and privacy issues in internet-of-things, IEEE Internet

Things J. 4 (2017) 1250−1258, https://doi.org/10.1109/JIOT.2017.2694844.

[5] N. Rahim, Bibliometric analysis of cyber threat and cyber attack literature: exploring the higher education context, in: Cybersecurity Threats with New Perspectives, IntechOpen, London, United Kingdom, 2021: pp. 142−157, https://doi.org/10.5772/intechopen. 98038.

[6] M. Husák, V. Bartoš, P. Sokol, A. Gajdoš, Predictive methods in cyber defence: current experience and research challenges, Future Generat Comput Syst. 115 (2021) 517−530, https://doi.org/10.1016/j.future.2020. 10.006.

[7] K. Kim, J.S. Kim, S. Jeong, J.H. Park, H.K. Kim, Cybersecurity for autonomous vehicles: review of attacks and defence, Comput Secur. 103 (2021) 102150−102177, https://doi.org/10.1016/j.cose.2020. 102150.

[8] C. Phillips, L.P. Swiler, A graph-based system for network-vulnerability analysis, in: Proc Workshop New Secur, Para., 1998: pp. 71−79, https://doi.org/10.1145/310889.310919.

[9] X. Zhang, Q. Wang, X. Wang, R. Zhang, Attack path analysis of power monitoring system based on attack graph, IOP Conf Ser Earth Environ Sci. 645 (2021) 12064−12070, https://doi.org/10.1088/1755-1315/645/1/012064.

[10] L. Wang, T. Islam, T. Long, A. Singhal, S. Jajodia, An attack graph-based probabilistic security metric, 5094, Artificial Intelligence, Springer, 2008: pp. 283−296, https://doi.org/10.1007/978-3-540-70567-3_22.

[11] K.P. Grammatikakis, I. Koufos, N. Kolokotronis, C. Vassilakis, S. Shiaeles, Understanding and mitigating banking trojans: from Zeus to emotet, in: Proc Int Conf Cyber Secur and Resilience, IEEE, 2021: pp. 121−128, https://doi.org/10.1109/CSR51186.2021.9527960.

[12] K. Kaynar, A taxonomy for attack graph generation and usage in network security, J Inf Secur Appl. 29 (2016) 27−56, https://doi.org/10.1016/j.jisa.2016.02.001.

[13] Z.J. Al-Araji, S.S.S. Ahmed, R.S. Abdullah, A.A. Mutlag, H.A. A. Raheem, S.R.H. Basri, Attack graph reachability: concept, analysis, challenges and issues, Netw Secur. 2021 (2021) 13−19, https://doi.org/10.1016/S1353-4858(21)00065-9.

[14] A. Sabur, A. Chowdhary, D. Huang, A. Alshamrani, Toward scalable graph-based security analysis for cloud networks, Comput Network. 206 (2022) 108795−108815, https://doi.org/10.1016/j.comnet.2022.108795.

[15] H. Li, Y. Wang, Y. Cao, Searching forward complete attack graph generation algorithm based on hypergraph partitioning, Procedia Comput Sci. 107 (2017) 27−38, https://doi.org/10.1016/j.procs.2017.03. 052.

[16] Y. Chen, K. Lv, C. Hu, Optimal attack path generation based on supervised Kohonen neural network, in: Int Conf Netw Sys Secur, Springer, 2017: pp. 399−412, https://doi.org/10.1016/j.jnca.2008.06.001.

[17] Z. Yichao, Z. Tianyang, G. Xiaoyue, W. Qingxian, An improved attack path discovery algorithm through compact graph planning, IEEE Access. 7 (2019) 59346−59356, https://doi.org/10.1109/ACCESS.2019. 2915091.

[18] G. Frances, H. Geffner, Modeling and computation in planning: better heuristics from more expressive languages, Proc ICAPS (25) (2015) 70−78, https://doi.org/10.17/936-98-37.

[19] X. Guan, Y. Ma, Y. Hua, An attack intention recognition method based on evaluation index system of electric power information system, in: Proc Inform Technol Netw Electron Auto Contr Conf, IEEE, 2017: pp. 1544−1548, https://doi.org/10.1109/ITNEC.2017. 8285053.

[20] M. Ibrahim, A. Alsheikh, Automatic hybrid attack graph (AHAG) generation for complex engineering systems, Processes. 7 (2019) 1−15, https://doi.org/10.3390/pr7110787.

[21] W. Shuo, G. Tang, G. Kou, Y. Chao, An attack graph generation method based on heuristic searching strategy, 2nd Int Conf Comput Comm (2016) 1180−1185, https://doi.org/10.1109/CompComm.2016.7924891.

[22] J.B. Hong, D.S. Kim, Towards scalable security analysis using multi-layered security models, J Netw Comput Appl. 75 (2016) 156−168, https://doi.org/10.1016/j.jnca. 2016.08.024.

[23] T. Li, H. Zhang, J. Wang, N. Wang, Research for modelling network security based on attack-defence grapy of state reduction, IET Conf Publ (2015) 52−56, https://doi.org/10.1049/cp.2015.0805.

[24] K. Cook, T. Shaw, P.J. Hawrylak, J. Hale, Scalable attack graph generation, in: Proc Annu Cyber and Inform Secur Res Conf, 2016: pp. 1−4, https://doi.org/10.1145/2897795.2897821.

[25] Y. Zhang, B. Wang, C. Wu, X. Wei, Z. Wang, G. Yin, Attack graph-based quantitative assessment for industrial control system security, in: Proc Chinese Automat, Congr. CAC, 2020: pp. 1748−1753, https://doi.org/10.1109/CAC51589.2020.9327842.

[26] T. Hughes, O. Sheyner, Attack scenario graphs for computer network threat analysis and prediction, Complexity. 9 (2003) 15−18, https://doi.org/10.1002/cplx.20001.

[27] M. Husák, J. Komárková, E. Bou-Harb, P. Čeleda, Survey of attack projection, prediction, and forecasting in cyber security, IEEE Commun Surv Tutor. 21 (2018) 640−660, https://doi.org/10.1109/COMST.2018. 2871866.

[28] L. Wang, A. Liu, S. Jajodia, Using attack graphs for correlating, hypothesising, and predicting intrusion alerts, Comput Commun. 29 (2006) 2917−2933, https://doi.org/10.1016/j.comcom.2006.04.001.

[29] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, D. Huang, NICE: network intrusion detection and countermeasure selection in virtual network systems, IEEE Trans Dependable Secure Comput. 10 (2013) 198−211, https://doi.org/10.1109/TDSC.2013.8.

[30] I. Kotenko, A. Chechulin, A cyber attack modeling and impact assessment framework, in: Int Conf Cyber Confl CYCON, IEEE, 2013: pp. 1−24.

[31] S. Fayyad, C. Meinel, Attack scenario prediction methodology, in: Proc 10th Int Conf Inf Technol: New Generations, 2013: pp. 53−59, https://doi.org/10.1109/ITNG.2013.16.

[32] A.A. Ramaki, M. Khosravi-Farmad, A.G. Bafghi, Real-time alert correlation and prediction using Bayesian networks, in: Proc 12th Int ISC Conf Inf Secur Cryptol, 2015: pp. 98−103, https://doi.org/10.1109/ISCISC.2015.7387905.

[33] H. Orojloo, M. Abdollahi Azgomi, Predicting the behaviour of attackers and the consequences of attacks against cyber-physical systems, Secur Commun Network. 9 (2016) 6111−6136, https://doi.org/10.1002/sec.1761.

[34] N. Polatidis, E. Pimenidis, M. Pavlidis, S. Papastergiou, H. Mouratidis, From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks, Evol Syst. 11 (2018) 1−12, https://doi.org/10.1007/s12530-018-9234-z.

[35] Z.J. Al-Araji, S.S.S. Ahmad, R.S. Abdullah, Propose vulnerability metrics to measure network secure using attack graph, Int J Adv Comput Sci Appl. 12 (2021) 51−58, https://doi.org/10.14569/IJACSA.2021.0120508.

[36] G. Chartrand, P. Zhang, A first course in graph theory, Dover Publications, Inc., New York, USA, 2013.

[37] S. Schiaffino, A. Amandi, Polite personal agents, IEEE Intell Syst. 21 (2006) 12−19, https://doi.org/10.1109/MIS.2006.15.

[38] S. Jha, O. Sheyner, J. Wing, Two formal analyses of attack graphs, Proc Comput Secur Founda (2002) 49−63, https://doi.org/10.1109/CSFW.2002. 1021806.

[39] K. Kaynar, F. Sivrikaya, Distributed attack graph generation, IEEE Trans Dependable Secure Comput. 13 (2015) 519−532, https://doi.org/10.1109/TDSC.2015.2423682.

[40] G. Liu, K.G. Ramakrishnan, A *Prune: an algorithm for finding K shortest paths subject to multiple constraints, Proc INFOCOM, IEEE. 2 (2001) 743−749, https://doi.org/10.1109/infcom.2001.916263.

[41] J. Rose, M. Swann, G. Bendiab, S. Shiaeles, N. Kolokotronis, 913 Malicious Network Traffic PCAPs and Binary Visualisation Images Dataset, IEEE Dataport, 2021. https://ieee-dataport.org/open-access/913- malicious- network- traffic- pcaps- and -binary-visualisation-images-dataset. (Accessed 23 April 2022).

[42] M. Swann, J. Rose, G. Bendiab, S. Shiaeles, N. Savage, Tools for Network Traffic Generation - A Quantitative

Comparison, 2020: pp. 24−29, https://doi.org/10.20533/worldcis.2020.0003. ArXiv Preprint ArXiv: 2109.02760.

[43] R. Panigrahi, S. Borah, A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems, Int J Eng Technol. 7 (2018) 479−482, https://doi.org/10.1201/136.2018.3.

[44] D. Stiawan, M.Y. Bin Idris, A.M. Bamhdi, R. Budiarto, others, CICIDS-2017 dataset feature analysis with information gain for anomaly detection, IEEE Access. 8 (2020) 132911−132921, https://doi.org/10.1109/ACCESS.2020.3009843.

[45] T.A. Alhaj, M.M. Siraj, A. Zainal, H.T. Elshoush, F. Elhaj, Feature selection using information gain for improved structural-based alert correlation, PLoS One. 11 (2016) e0166017−e0166034, https://doi.org/10.1371/journal.pone.0166017.

[46] P. Bereziński, B. Jasiul, M. Szpyrka, An entropy-based network anomaly detection method, Entropy. 17 (2015) 2367−2408, https://doi.org/10.3390/e17042367.

[47] J. Han, M. Kamber, J. Pei, Data mining concepts and techniques, in: The Morgan Kaufmann Series in Data Management Systems, 3rd ed., Morgan Kaufmann, Singapore, 2011: pp. 83−124, https://doi.org/10.1016/C2009-0-61819-5.

[48] M.C. Belavagi, B. Muniyal, Performance evaluation of supervised machine learning algorithms for intrusion detection, Procedia Comput Sci. 89 (2016) 117−123, https://doi.org/10.1016/j.procs.2016.06.016.

[49] H.S. Lallie, K. Debattista, J. Bal, A review of attack graph and attack tree visual syntax in cyber security, Comput Sci Rev. 35 (2020) 100219−100260, https://doi.org/10.1016/j.cosrev.2019.100219.

[50] Geeksforgeeks, Find all paths between two nodes, print all paths from a given source to a destination. https://www.geeksforgeeks.org/find- paths- given-source-destination/, 2022. (Accessed 6 April 2022).

[51] M. Li, P. Hawrylak, J. Hale, Concurrency strategies for attack graph generation, in: Proc - 1st Int Conf Data Intell Secur ICDIS, IEEE, 2019: pp. 174−179, https://doi.org/10.1109/ICDIS.2019.00033.

[52] Y. Feng, G. Sun, Z. Liu, C. Wu, X. Zhu, Z. Wang, B. Wang, Attack graph generation and visualization for industrial control network, Chin Control Conf CCC (2020) 7655−7660, https://doi.org/10.23919/CCC50068.2020.9189450.