El Cezeri
Journal of Science and Engineering
ISSN: 2148-3736

Research Article

Implement Edge Pruning to Enhance Attack Graph Generation Using the Naïve Approach Algorithm

Zaid J. Al-Araji[1a], Ammar Awad Mutlag[1b], Sharifah Sakinah Syed Ahmad[1c]
[1] Technical Computer Engineering Department, Al-Hadba University, Mosul, Iraq
[2] Ministry of Education/General Directorate of Curricula, Pure Science Department, Baghdad 10065, Iraq
[3] Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia

zaid.jm@hcu.edu.iq

DOI : 10.31202/ecjse.1375755
Received: 13.10.2023 Accepted: 18.01.2024
How to cite this article:
Zaid J. Al-Araji, Ammar Awad Mutlag, Sharifah Sakinah Syed Ahmad, " Implement Edge Pruning to Enhance Attack Graph Generation Using the Naïve Approach Algorithm ", El-Cezeri Journal of Science and Engineering, Vol: 11, Iss:3, (2024), pp.(298-306).
ORCID: [a]0000-0003-1768-7569; [b]0000-0002-4966-0232; [c]0000-0002-3803-4578.

Abstract : The use of network technologies has increased in recent years. Although the network is beneficial for individuals to work and live in, it does have security challenges that should be rectified. One of these issues is cyberattacks. The attack surface for hackers is growing as more devices are linked to the internet. The next-generation cyber defence concentrating on predictive analysis seems more proactive than existing technologies based on intrusion detection. Recently, many approaches have been proposed to detect and predict attacks; one of these approaches is attack graphs. The main reason for designing the attack graph is to predict the attack as well as to predict the attack's next step in the network. The attack graph depicts the many paths an attacker may attempt to get around a security policy by leveraging interdependencies between disclosed vulnerabilities. The attack graph is categorized into three sections: generation, analysis, and use of attack graph. However, current attack graphs are suffering from a few issues. Scalability is the main issue the attack graph generation is facing. The reason for this issue is that the increase in the usage of devices connected to the network leads to increased vulnerabilities in the network, which leads to an increment in the complexity as well as generation time of the attack graph. For this issue, this study proposes using the naïve approach prune algorithm and using Personal agents to reduce the reachability time in calculating between the nodes and to remove unnecessary edges, minimizing the attack graph's complexity. For the results, the proposed attack graph performs better than the existing attack graph by using a naïve approach and a personal agent. The proposed attack graph reduced the generation time by 20 percent and the attack graph complexity.

Keywords : Attack Graph, Fog computing, Cloud computing, Edge Computing, Edge Pruning

1 Introduction

The rapid development of telecommunications technology has profoundly altered people's lives. The networks have been actively involved in each life aspect to break the limits of space and time that have greatly enabled humanity. As more people invest in cyberspace to execute their everyday tasks, this trend reveals that people worldwide are utilizing internet capabilities. Security problems have grown as a result of the Internet's continuous rise in connections [1] The network becomes more vulnerable as a result of the rising utilization of the devices that are connected to the internet. Vulnerabilities are flaws in the design of a system that allows an outsider to execute commands, obtain unauthorized data, and launch numerous attacks [2] [3] [4]. According to the National Vulnerability Database (NVD), the number of vulnerabilities was over 18362 in 2020. Also, this is more significant compared to previous years (17,382 in 2019 and 17,252 in 2018) [5]. This increase in vulnerabilities through the years has led to increased attacks on organizations and individual networks [6]. However, these best security measures do not consider application, network service-based, or operating system vulnerabilities. Attackers abuse these vulnerabilities and get legal access to the network resources, evading access control and authentication measures, owing to the evolution of sophisticated hacking tools. An attack graph is a tool that provides a concise description of various attack scenarios relevant to a network [7] [8] [9] [10]. According to [11] [12] [13] [14], the generation of an attack graph could be divided into four phases: reachability analysis, modelling, core building, and analysis. The reachability analysis is the first phase. To summarize, this phase looks into the reachability circumstances within the target network, showing if two specific hosts can connect. The second step is an attack graph model, which can be thought of as an attack template that specifies the components of various attacks, the conditions (acquired or necessary attacker capabilities), and the relationships between the requirements. The rationale for several attacks is

Table 1: Attack graph generation previous work

Author	Strength	Limitation
[16]	The generation time is stable with Different numbers of vulnerabilities	- The complexity calculation is not considered - Consider the attack path discovery only in the analysis. - The starting node is assigned manually.
[19]	The attack graph generation has been done automatically.	- Generation time and complexity are not calculated. - The attack graph does not support large-scale networks.
[20]	Use Hadoop MapReduce to enhance the scalability	- The experiment used only eight machines
[21]	Use the partition graph to generate the attack graph faster	- Create an attack graph statically - The generating algorithm's complexity is not measured
[22]	Used two metrics to find the optimal path	- The complexity of the generation algorithm is not considered.
[23]	Relevant in the context of large-scale complex networks	- Calculating the time and algorithm complexity is not considered.
[24]	Use attack probability	- Time and complexity are not considered
[25]	Propose an algorithm to divide the services into segments	- The attack graph is generated based on connections, vulnerabilities, and services. - The attack graph analysis is not considered

defined by an attack template [15]. The third phase, known as attack graph core building, deals with the fundamental technique that generates the attack graphs. The fourth phase is attack graph analysis, which refers to attack path discovery, which means predicting the attack path from the initial node to the target node [12]. The attack graph represents the possible ways in which an attacker might violate a security policy [10], [16]. An attack graph may be created using known network vulnerabilities as well as network configuration details [17]. The attack graph shows how an attacker might utilize vulnerability dependencies to circumvent a security policy. However, the attack graph performance still suffers from different issues, the generation time is the main challenge of the attack graph [18]. Based on this, this paper proposed a naïve approach algorithm to prune the unnecessary edge in the attack graph which reduces the complexity of the attack graph, and used a personal agent to calculate the reachability of each node individually which leads to reducing the reachability calculation time. The rest of the paper is organized by, section 2 is the related work, section 3 explains the proposed model, section 4 shows the experiments, section 5 explains the results, and section 6 is the conclusion.

2 Related Work

Many attack graphs have recently been developed utilizing various methodologies and methods to improve attack graphs. The most current attack graphs will be discussed in this section as shown in Table 1.

[19] introduced Derivation of Attack Graphs Generation from Security Requirements (DAGGER), a novel attack graph-building approach that employs a comprehensive collection of threat models to automatically evaluate the security posture of heterogeneous and complex (both new and established) cyber-physical system designs. To increase readability, [20] proposes a graph segmentation algorithm for breaking apart a complex attack graph into several subgraphs while maintaining the original structure. The authors also suggest a distributed attack graph segmentation method based on Spark GraphX and a distributed attack graph synthesis method based on Hadoop MapReduce to handle enormous amounts of alert data. [21] proposed a hypergraph partitioning-based searching forward complete attack graph design strategy to assure that each computing agent parallels computational load is balancing. Furthermore, calculating the whole attack graph in the opposite order from the vulnerability nodes to the attacker reduces calculation duplication in real-time attack graph development, adds to real-time network vulnerability analysis, and ensures computational accuracy and efficiency. A supervised Kohonen neural network-based attack graph-generating algorithm is suggested by [22]. Using this strategy, the authors can predict the attack success rate and status types if attackers successfully exploit vulnerabilities. [16] presented a "compact planning graph"-based attack path discovery algorithm that may use goal-oriented information from penetration testing to sever recurring attack path branches [26]. In a complex network context, [23] recommended a graph database-based approach to network vulnerability assessment and its usefulness in addressing state explosion, too many attributes and high computational burden issues were investigated. [24] Proposed an attack path prediction method to find the optimal attack path based on the attack graph. The authors calculate the attack probability by attack cost and vulnerability value. A network segmentation-based scalable security state (S3) framework is presented by [25]. The framework divides the large-scale network region into smaller, more manageable units using the well-

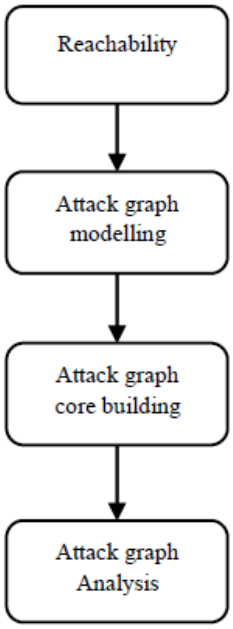

Figure 1: Attack graph generation steps

known divide-and-conquer strategy. To divide the system into segments based on how similar the services are to one another, this study used a well-known segmentation technique created from the K-means clustering algorithm. The segments are kept isolated by a distributed firewall (DFW), which also blocks lateral attacker movement that might compromise them. In conclusion, even though several techniques have been employed to enhance attack graph development, there are still certain problems with attack graphs, particularly scalability, which affects the attack graph generation time.

3 Proposed Model

The generation of the attack graph in this work can be divided into three steps which are reachability analysis, attack graph modelling, attack graph core building and attack graph analysis as the following as shown in Figure 1. In the following, each step will be explained in detail.

3.1 Attack Graph Reachability

Calculating reachability is a time-consuming and difficult operation. It discovers paths between target and source nodes using network topology information. All the filtering devices in the networks must have their rulesets imported and simulated. There are several methods for calculating reachability. The reachability between nodes in the network will be calculated using a matrix and multi-graph in this paper. Here, the matrix is employed to determine the connection between the nodes using the IP addresses, while the multi-graph is used to visualize the network topology and the weaknesses in the network.

A multi-graph refers to a graph that allows for multiple edges (also known as parallel edges) [15], [27]. In other words, they are known as edges that possess similar end nodes. As a result, one node can connect to another node with one or more edges. The goal of calculating the reachability using the multi-graph is to reach every IP address in the network and show all the weaknesses between the nodes.

The accessibility of software programs placed on the target network is determined by reachability. The reachability conditions are determined by firewall filtering rules, trust relationships, router access control lists, and software programs. To develop our reachability multi-graph, all of these aspects will be considered.

A personal agent gathers these elements from the nodes, determines the node's reachability, and sends the results to the administrator server, which is in charge of producing the attack graph. Each personal agent is in charge of gathering information to calculate reachability; the personal agent is also responsible for the update to reduce the time complexity. For numerous software applications, the administrator server will be in charge of identifying information source usages, vulnerability exploits, and attacker privileges.

The graph-vertex represents the nodes in the network. On the other hand, a graph edge denotes a group of source and target software programs where the applications' source may access the target applications if certain requirements are met. The circumstances are saved in the edge, enabling direct reachability between software applications. Port numbers, network protocols, user passwords, and many others are examples of such situations.

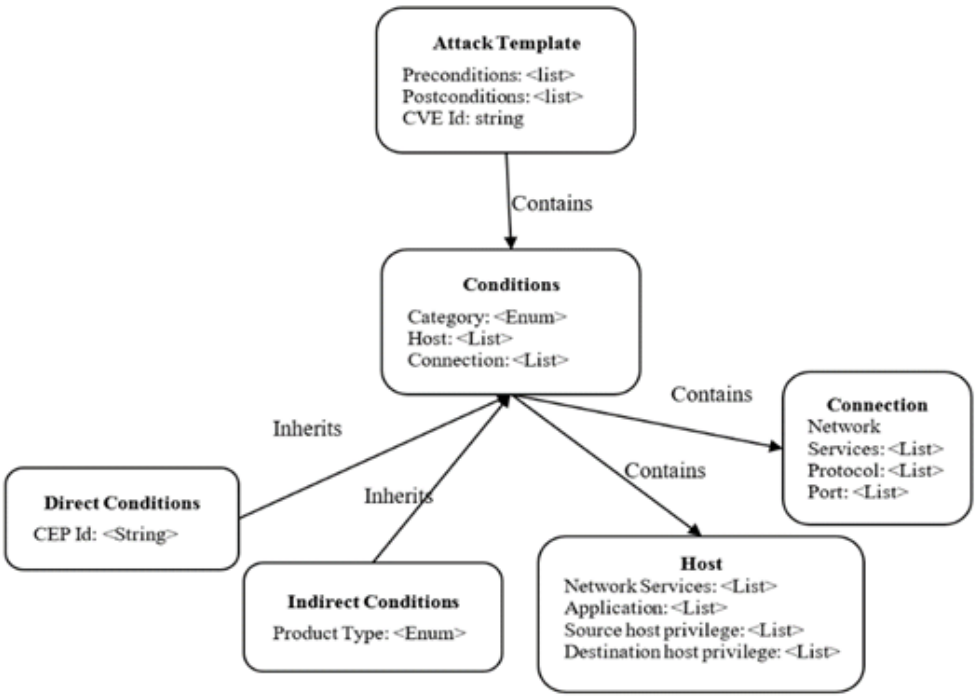

Figure 2: Attack Template

3.2 Attack Graph Modeling

Two components of attack graph modelling are figuring out the structure of attack graphs and modelling attack templates. The modelling of the attack template includes the pre- and post-conditions for vulnerabilities. It also contains a way for determining these requirements for particular vulnerabilities using weak databases and public vulnerability. Note that choosing the types of edges and nodes in an attack graph determines the attack graph structure. Network modelling aims to find the best depiction for network assets (such as hardware/software applications operating on network hosts). Each of these sub-parts is discussed in particular in the subsequent sub-sections.

3.2.1 Attack Template Modelling

The attack template model is derived from the vulnerability data in CVE and NVD, which uses the formalization method to represent the attack template factors. It was developed by [28], who described a four-part attack template: network preconditions, attacker preconditions, network postconditions, and attacker postconditions. In this work, the same attack template as in [21] will be employed. Figure 2 depicts the attack template model employed in our system, followed by the formal descriptions of its elements.

Definition 1. A condition refers to a type of right obtained in a software program. It refers to a two-element tuple, $< Category, Host >$, with *Category* denoting the gains on the programme and *Host* denoting the software application location where the right is earned. The attacker's and victim's software applications are used to ascertain the location.

Definition 2. In a condition tuple, a direct condition provides an additional component. The *CPE* product identifier of the software program in which the condition is gained is represented by this element, which is *CPEId*. The *CPEId* is provided by NVD.

Definition 3. A condition tuple with an indirect condition has an extra element. *ProductType* is an element that identifies the sort of software program based on the condition category. The *producttype* can be a mail server, mail client, database server application, and many others. Indicate the technology class of software application.

Definition 4. This thesis considers a vulnerability in a specific CVE entry in the CVE database. $< CVEId, Preconditions, Postcondition$ is a three-element tuple that represents it. The *CVEId* element stores the unique CVE identification for each vulnerability. It possesses a list of preconditions that indicate which attacker privileges are necessary to exploit the vulnerability.

3.2.2 Attack Graph Structure

The attack graph structure determines the generated attack graphs' edges and nodes. As illustrated in Figure 3, the attack graph structure suggested by [15] will be employed.

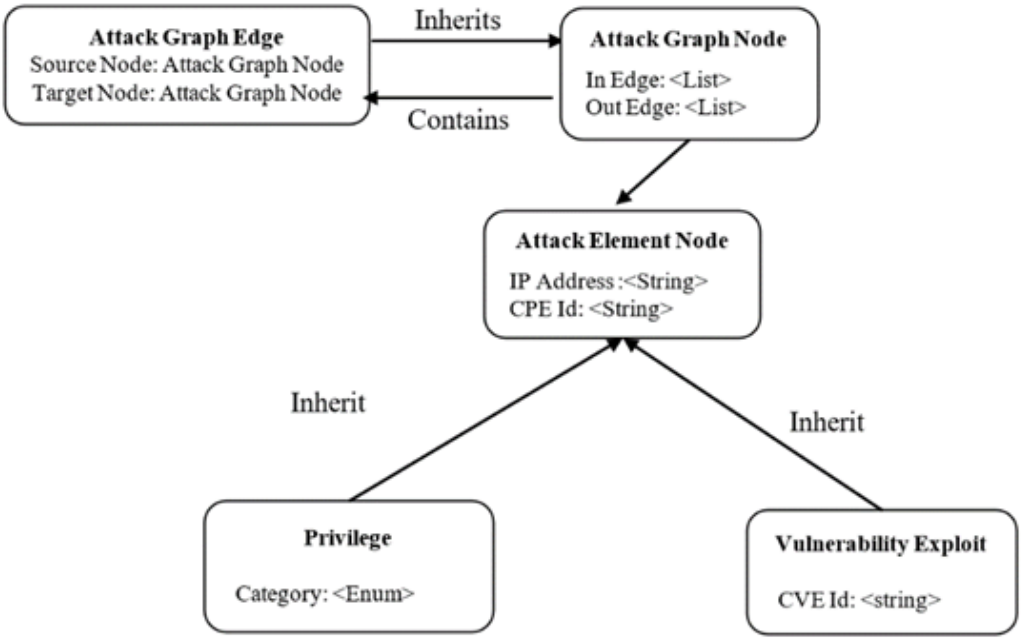

Figure 3: Attack Graph Structure

```
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);
                        Vul_list = check_vulnerabilities(M.vulnerabilities, n.vulnerabilities)
                        If (Vul_list ≠ null)
                                M.P++;
                                addedge(node, Vul_list, n);
                        Else
                                addedge(node, privilege, 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
```

Figure 4: Attack graph generation

Definition 5. A privilege node resembles an attacker privilege on a software application on a network host Definition 6. In an attack graph, a vulnerability conjunction node represents a conjunction connector for several vulnerability nodes. Definition 7. A software application vulnerability on a network host is indicated by a vulnerability node.

3.3 Attack Graph Core Building

The administrator will generate the attack graph when data is collected from every node using a personal agent, as shown in Figure 4. The algorithm starts by extracting each node's information in the network, comprising the vulnerabilities, reachability,

```
1-  Input
    G=(V,E)
    (s, t),                          //Node pair with source and target
    edges_list, app_list, vul_list, AHE
2-  Process
    AHE ← sort (edges_list);
    While (AHE is not empty)
            (s, t) ← first edge in AHE
            Remove first edge from AHE
            sourceapp_list [] ← app_list[s]
            distapp_list [] ← app_list[t]
            vul_list = findvulnerabilities (sourceapp_list, distapp_list)
            sourcevul_list [] ← vul_list [s]
            distvul_list [] ← vul_list [t]
            gainprivileges = findgainprivileges (s, t)
            if (gainprivileges !=0)
                    graph.addedge (s, gainprivileges)
                    graph.addedge (gainprivileges, t)
                    goto 2;
            elseif (vul_list !=0)
                    graph.addedge (s, vul_list)
                    graph.addedge (vul_list, t)
                    goto 2;
            else
                    goto 2;
            endif
    end while
3-  Output
    Prune Attack Graph
```

Figure 5: Naïve approach pruning algorithm

application list, privileges, and connection information. The next step is checking the mutual vulnerability between the node and the reachability list for the node. If the reachability nodes list and the node are related by more than one vulnerability, the node will be classified as a conjunction node. If a vulnerability exists between the list of reachability nodes and the node, the node is assigned as a vulnerability node. If there is no vulnerability connected to the node, the node will be assigned as a privileged node.

3.3.1 Naïve Approach

The naïve approach pruning algorithm was presented first by [29]. A naïve approach was proposed to simplify the network visualization. Naïve approach algorithm is used to prune some of the edges in the graph to minimize the loss of connectivity.

In this work, a naïve approach will be used to prune unnecessary edges from the attack graph to reduce the complexity of the generation. The definitions of the algorithm will be the following:

Definition 8: Let a network be expressed using a graph $G = (V, E)$, in which V denotes the set of nodes and E denotes the set of links.

Definition 9: let the e -> E be a pair of $e = s, t$ of nodes s and t -> V, as in Equation (1)

$$e- > E, E = e(s_0, t_0), e(s_1, t_1), \ldots, e(s_n, t_n). \tag{1}$$

Definition 10: App edge: Suppose app m is denoted as an edge between nodes s and t. Here, m resembles an application of joint application list set M between nodes s and t , as in Equation (2)

$$m- > M, M(s, t) = app_1, app_2, \ldots.., app_n. \tag{2}$$

The algorithm's first step is to get the full attack graph with V, which is nodes and E, which is edges, also gets the list of vulnerabilities in the network vul_list, application list app_list for all nodes in the full attack graph, as in Figure 5.

In the beginning, the algorithm started to sort all the edges ascendingly based on the weights of the edge. Then add them to AHE_list which means (acceptable head edge) that it contains all the edges after sorting them.

After sorting the edges, the first edge in AHE_list will be taken to the test. The edge will be removed from the AHE_list then the edge will be checked by the vulnerabilities in each node. First, it will be checked if there is a gain privilege between these nodes which means there is an impropriate behaviour in one of these nodes which could be an attacker; then add an edge between the nodes and the gain privileges. Secondly, if there are no gain privileges, check the vulnerabilities between these nodes that might be used by the attacker to attack the node or to move from one node to another, then add an edge between the nodes and the highest vulnerabilities score between all the vulnerabilities in the edge. Lastly, if there are no privileges or vulnerabilities between these nodes, go to another edge.

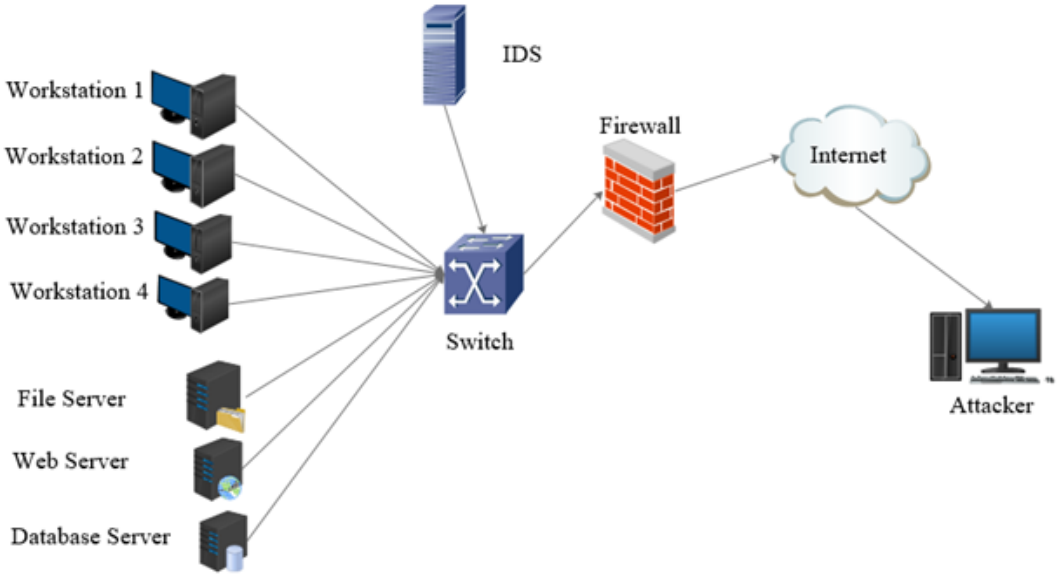

Figure 6: Network Topology

4 Experiment

In order to compare our approach to other attack path-finding techniques, the experiment was run on a real enterprise network. The network topology shown in Figure 6 was used to test the created attack graph; due to the network's small size and low number of nodes, it is referred to as a small-scale network.

Three servers make up the network: a file server, a database server, and a web server. Additionally, it features an IDS device and four workstations, of which two are running Windows 10 and the other two are Windows 7. A perimeter firewall is present. In this network configuration, network connectivity is governed by the following firewall rules:

- The webserver and workstations are connected to the network in both directions.
- Workstations 1, 2, 3, and 4 are connected in both directions.
- The attacker's external host is connected to the internet and has access to the web server's HTTP protocol and HTTP port.
- Workstations 3 and 4 are in communication with one another in both directions.
- The four workstations and file servers can connect using the NFS protocol and NFS port.
- The four workstations and file servers have connections to the internet via the HTTP protocol, while workstation 4 has access to the network's database server. The studies used to assess the generating attack graph in this setting included a core i7 3.40 GHz processor, 32 GB of RAM, Windows 11 as the operating system, and Microsoft Visual Studio 2022 for coding.

5 Results

As a result, there are no missing data when the attack graph is updated if any information has changed, according to the results of employing a personal agent to gather the data and communicate it to the administrator. By using a naive technique of pruning, the attack graph's complexity and generation time are both decreased. As a result, after applying the naive technique prune, the running time for the network design in Figure 5.1 is 0.247 seconds. These findings motivate us to test additional attack graph components, as well as larger nodes and servers (large-scale networks), to evaluate the complexity and running time of the graph using the personal agent and naive approach prune algorithm.

The first factor is the attack graph generation time. The experiment compares the running time in different stages of attack graph generation. The first stage is the full graph without applying any pruning algorithm and parallel process, while the second stage applies a parallel process to execute each sub-network individually. The last stage of our work includes the naïve approach prune algorithm, a parallel process, as in Table 2. The results show the difference in each stage until reaching the last stage. The last stage records less generation time because it uses the parallel process and naïve approach, which reduces the number of edges used in the generation and reduces the attack graph's generation time.

Other algorithms are used to generate the attack graph, which are greedy algorithm, breadth-first search, and depth-first on the same number of nodes and a similar topology. These path-pruning algorithms were chosen to compare because these algorithms have been used to prune the attack graph in previous works. The results show that the naïve approach prune algorithm gets better results than other algorithms, as shown in Figure 7; the reason behind that is naïve approach is based on edge pruning while other algorithms are search algorithms, so they depend on path pruning. The path pruning algorithms require more time because they need to identify the connection between the nodes, determine all the paths then check each path individually; while

Table 2: Running time of the attack graph (seconds)

Nodes Number	Full Graph	Parallel Process	Our Work
8 nodes	12.604	5.973	1.232
50 nodes	27.489	13.187	5.173
100 nodes	62.480	27.875	13.579
250 nodes	113.631	56.319	39.496
400 nodes	201.170	117.642	91.670

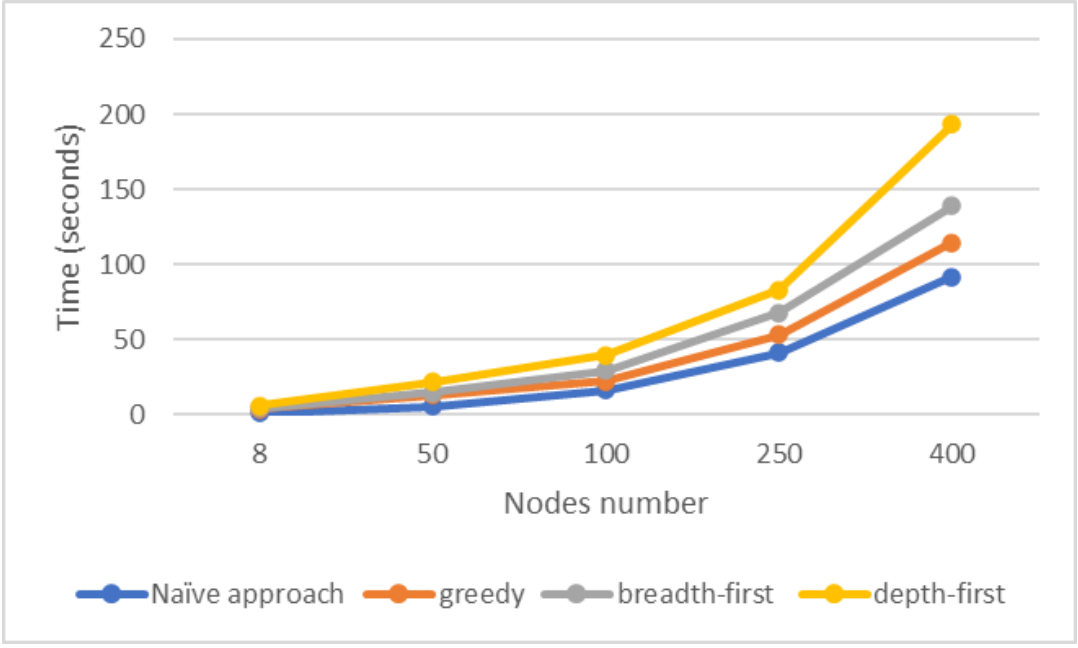

Figure 7: Comparison results between the Naïve approach and other algorithms

each pruning algorithm requires only the connection between the nodes and then directly checks the edge, also can check more than one edge at the same time.

As stated in Table 3, a further comparison with earlier research has been conducted. When comparing the outcomes, it can be seen that our work is more efficient than others because [15] relies on multiple agents to compute and produce the subgraphs, which are then "combined in the administrator to produce the full graph." While Li et al. [30] used the queue size to indicate the number of threats and reduce creation time, merging the sub-graphs is a problem because it takes a while to construct the whole graph. While Feng et al. [10] rely on a reduction in the number of vulnerabilities in each node to ensure that the attack graph won't be larger and that it won't increase generation time, the generating time will rise as the number of vulnerabilities increases because the attack graph is intended to reflect all of the network's vulnerabilities.

The complexity of the attack graph is measured next. The attack graph represents the connection between the nodes; however, the attack graph generally depends on the reachability calculation, so the general complexity in the worst scenario is $O(N^2)$ where N is the number of nodes; in this thesis, a personal agent is used to calculate the reachability for each node, so the complexity of the attack graph will be divided to between each host, so the reachability's complexity is O(N), while combining each reachability by the administrator which is O(N*logN), so the complexity is O(N+(N*logN)). Also, the naive approach is used to prune unnecessary edges in the attack graph; the naive approach complexity is $O(E)$, where E denotes the number of the edges that must be checked; however, the network is divided into subnetworks so that each subnetwork will generate the attack graph individually then send it to the administrator, so the complexity for the naive approach become $O(E/S)$ where S is the number for subnetworks. However, the attack graphs must be merged to get the full attack graph, so the merged graph complexity is O(N*log N), so basically, the naive approach complexity will be $O((E/S) + (N * logN))$. In the end, the attack graph complexity will be $O(N + 2(N * logN) + (E/S))$, so it will be $O(N + (N * logN) + (E/S))$.

Table 3: Running time of the attack graph (seconds)

Network size	[15]	[30]	[10]	our work
8	4.102	9.350	5.616	1.232
50	9.830	13.610	11.381	5.173
100	14.670	21.583	16.620	13.579
250	47.38	52.172	49.521	39.496

6 Conclusion

Nowadays, networks have overgrown in both terms of complexity and size. However, with the widespread growth of network connectivity, the frequency of cyber-attacks on enterprises and government offices has risen dramatically, causing business interruptions and compromising the reputation and monetary stability of these organizations. To overcome with this issue, the researcher uses a graph model to predict, detect, and discover attack paths to represent the network, however, the attack graph generation is still having an issue of generation time. In this work, a naïve approach algorithm and personal agent are used to enhance the generation time by reducing the reachability calculations and pruning unnecessary edges in the graph. A personal agent gathers the information from the nodes, determines the node's reachability, and sends the results to the administrator server, which is in charge of producing the attack graph. Each personal agent is in charge of gathering information to calculate reachability; the personal agent is also responsible for the update to reduce the time complexity. While the naïve algorithm is used to prune the unnecessary edge that will reduce the probabilities of the paths that might be used by the attacker. The experiment shows that the proposed model reduced the generation by 20 percent. In future work, use more attack graph reachability content to increase the information that used in the attack graph which increases the ability to detect and predict the attacks. Also, propose a new pruning algorithm with more details to reduce the generation time and the complexity of the attack graph.

Authors' Contributions

In this study, all authors studied in creating idea, designed the study and wrote up the article.

Competing Interests

The authors declare that they have no conflict of interest.

References

[1] J. Jang-Jaccard and S. Nepal, "A survey of emerging threats in cybersecurity," *Journal of computer and system sciences*, vol. 80, no. 5, pp. 973–993, 2014.
[2] E. Bertino, L. Martino, F. Paci, A. Squicciarini, E. Bertino, L. D. Martino, F. Paci, and A. C. Squicciarini, "Web services threats, vulnerabilities, and countermeasures," *Security for web services and service-oriented architectures*, pp. 25–44, 2010.
[3] J. M. Kizza, W. Kizza, and Wheeler, *Guide to computer network security*, vol. 8. Springer, 2013.
[4] M. Abomhara and G. M. Køien, "Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks," *Journal of Cyber Security and Mobility*, pp. 65–88, 2015.
[5] A. O'driscoll, "Cyber security vulnerability statistics and facts of 2022," *Comparitech*, 2021.
[6] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017.
[7] J. Wang, "A generation method of attack graph based on evolutionary computation," in *2016 2nd International Conference on Advances in Energy, Environment and Chemical Engineering (AEECE 2016)*, pp. 28–31, Atlantis Press, 2016.
[8] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, and E. ı. Tatli, "Automated generation of attack graphs using nvd," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pp. 135–142, 2018.
[9] M. Ibrahim and A. Alsheikh, "Automatic hybrid attack graph (ahag) generation for complex engineering systems," *Processes*, vol. 7, no. 11, p. 787, 2019.
[10] Y. Feng, G. Sun, Z. Liu, C. Wu, X. Zhu, Z. Wang, and B. Wang, "Attack graph generation and visualization for industrial control network," in *2020 39th Chinese Control Conference (CCC)*, pp. 7655–7660, IEEE, 2020.
[11] K. Kaynar, "A taxonomy for attack graph generation and usage in network security," *Journal of Information Security and Applications*, vol. 29, pp. 27–56, 2016.
[12] M. Husák, J. Komárková, E. Bou-Harb, and P. Čeleda, "Survey of attack projection, prediction, and forecasting in cyber security," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 640–660, 2018.
[13] Z. J. Al-Araji, S. S. S. Ahmed, R. S. Abdullah, A. A. Mutlag, H. A. A. Raheem, and S. R. H. Basri, "Attack graph reachability: concept, analysis, challenges and issues," *Network Security*, vol. 2021, no. 6, pp. 13–19, 2021.
[14] Z. J. Al-Araji, S. Sakinah Syed Ahmad, H. M. Farhood, A. Awad Mutlag, and M. S. Al-Khaldee, "Attack graph-based security metrics: Concept, taxonomy, challenges and open issues," in *BIO Web of Conferences*, vol. 97, p. 00085, EDP Sciences, 2024.
[15] K. Kaynar and F. Sivrikaya, "Distributed attack graph generation," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 519–532, 2015.
[16] 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.
[17] Z. J. Al-Araji, S. S. S. Ahmad, and R. S. Abdullah, "Propose vulnerability metrics to measure network secure using attack graph," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 5, 2021.
[18] Z. Al-Araji, S. S. Syed Ahmad, R. S. Abdullah, *et al.*, "Attack prediction to enhance attack path discovery using improved attack graph," *Karbala International Journal of Modern Science*, vol. 8, no. 3, pp. 313–329, 2022.
[19] M. Moulin, E. Eyisi, D. M. Shila, and Q. Zhang, "Automatic construction of attack graphs in cyber physical systems using temporal logic," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pp. 933–938, IEEE, 2018.
[20] 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, p. 1026, 2020.
[21] 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.
[22] Y. Chen, K. Lv, and C. Hu, "Optimal attack path generation based on supervised kohonen neural network," in *Network and System Security: 11th International Conference, NSS 2017, Helsinki, Finland, August 21–23, 2017, Proceedings 11*, pp. 399–412, Springer, 2017.
[23] B. Yuan, Z. Pan, F. Shi, and Z. Li, "An attack path generation methods based on graph database," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, vol. 1, pp. 1905–1910, IEEE, 2020.
[24] P. Sun, H. Zhang, and C. Li, "Attack path prediction based on bayesian game model," in *Journal of Physics: Conference Series*, vol. 1955, p. 012098, IOP Publishing, 2021.
[25] A. Sabur, A. Chowdhary, D. Huang, and A. Alshamrani, "Toward scalable graph-based security analysis for cloud networks," *Computer Networks*, vol. 206, p. 108795, 2022.

[26] G. Frances and H. Geffner, ''Modeling and computation in planning: Better heuristics from more expressive languages,'' in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, pp. 70–78, 2015.
[27] G. Chartrand and P. Zhang, *A first course in graph theory*. Courier Corporation, 2013.
[28] S. Jha, O. Sheyner, and J. Wing, ''Two formal analyses of attack graphs,'' in *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*, pp. 49–63, IEEE, 2002.
[29] F. Zhou, S. Malher, and H. Toivonen, ''Network simplification with minimal loss of connectivity,'' in *2010 IEEE international conference on data mining*, pp. 659–668, IEEE, 2010.
[30] M. Li, P. Hawrylak, and J. Hale, ''Concurrency strategies for attack graph generation,'' in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*, pp. 174–179, IEEE, 2019.