International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 5, Issue No: 1, September 2019

Network Security Risk Assessment based on Critical Attack Graph Evaluation

Munya Saleh Ba Matraf[1], Mohamed Ali Saip[2]
[1]Department of Computer Engineering, Faculty of Engineering, Hadhramout University, Mukalla, Yemen.
[2]Human-Centered Computing Research Lab, School of Computing, Universiti Utara Malaysia, 06010 Sintok, Kedah, Malaysia

Abstract: As a network system dependencies increase, such systems are vulnerable due to some software misconfigurations, software flaws, and operating system service malfunctions and are exposed to various attacks. Network managers frequently rely on Attack Graphs to visually perform network systems security risk assessment. The Attack Graphs are very cumbersome to comprehend visually as they develop exponentially when network size rises or when vulnerabilities in a network increase in the number of hosts. This paper addresses the Attack Graph generation's scalability problems by leveraging the context of graph theory. MulVAL and Nessus scanner instruments were used respectively for Attack Graph generation and mapping of network data. A computational algorithm has been formulated which is capable of handling cycles. A valid path detection algorithm was also formulated to determine the most critical and valid paths required for the security risk assessment of the network purpose within an Attack Graph. The results showed that the Attack Graphs' proposed model reduces redundancy. This will help the security administrator make reasonable decisions on the network systems' security risk management.

Keywords: Graph, Cycles, Critical Path, Attack, Risk Management, Security

I. INTRODUCTION

In today's economy and national infrastructures, computer networks play significant roles. They are increasingly dependent on them in multiple areas of economic, financial, company, etc. Network systems are used for data communication with distinct software, services, and configuration operating together dependently. These systems are vulnerable and every year the vulnerabilities increase. Network security has, therefore, become one of the major challenges these days and needs to be evaluated to protect the network from any form of malicious intrusion. Prevention of intrusion is one of the efficient methods for improving network security and includes eliminating the network's cause of assaults or vulnerabilities. Prevention of intrusion begins with detecting possible attacks in the networks or having knowledge of how attackers can exploit the vulnerability of the network to break the security and obtain the goal of the attack before hardening the network.

An attacker can exploit multiple vulnerabilities in a network before achieving a specific goal, such as receiving root privilege on a server. Such attacks are known as multi-stage attacks. Attack Graph is a powerful tool that can provide information on the relationship between different vulnerabilities that the attacker can exploit and the privileges that the attacker gains as a result of exploiting those vulnerabilities. Attack Graph demonstrates the possible sequence or path of malicious actions that can be followed by an attacker to penetrate the network and obtain certain privileges. These vulnerabilities may lead to inappropriate network system configuration or the presence of a particular version of a software product.

Attack Graph takes into consideration the number of vulnerabilities on the target network, the conditions defining accessibility among vulnerable software instances, and the level of detail in vulnerability modeling. All of these influence the Attack Graph's size. This implies the larger the size of the network, the larger the Attack Graph size. The more vulnerabilities that exist on the target network, the bigger the Attack Graph's size. This means that it becomes harder to assess and automate their vulnerability to attack as the hosts in a network grow in size so that the Attack Graph becomes very big and complicated. Therefore Attack Graph's scalability problem is necessary and necessary in network systems for network hardening and network security risk management purposes.

II. RELATED WORKS

Attack Graph generation was first performed using the red team strategy; this was susceptible to mistakes and very tedious because it was based on the manual effort that was not appropriate for mild network size. Different methods were proposed to generate Attack Graphs automatically. Phillips and Swiller (1998) proposed the concept of Attack Graph, and Swiller et al (2001) presented a tool for generating Attack Graph. Attack templates were used in their model to represent generic steps in known attacks. Dacier et al (1996) proposed the notion of privilege graph, after which Ortalo et al (1999) illustrated the use of the privilege graph in network security. However, the Attack Graph could not be computed because even with only 13 vulnerabilities, they turned out to be too big.

NuSMV was used to calculate the multi-stage multi-host Attack Graph in Sheyner et al. (2002) and Sheyner (2004), a model checker. Ammann et al. (2002) used the monotonicity assumption to address the Attack Graph-related scalability issue and was able to successfully reduce the polynomial computational cost. Jajodia et al. (2005) used the algorithm described by Ammann et al. (2002) to implement an embedded, topological strategy to vulnerability assessment. Topological Vulnerability Analysis (TVA) was called this approach.

In Noel and Jajodia's work (2004), the various parts of the exploit-dependent Attack Graph generated by TVA in Jajodia et al. (2005) have collapsed to make visual understanding more interactive. Ammann et al. (2005) used an algorithm to calculate the suboptimal attack route between each pair of hosts in a network. This job could discover the highest privilege that each host can gain as the attacker exploits the network's vulnerabilities. Man et al. (2008) proposed a breadth-first-generation algorithm by adding the attack step and probability of achievement to restrict the graph scale. Bhattacharya et al. (2008) suggested an algorithm for the identification of a generic attack route and showed that the routes of the attack are scalable. Tang et al. (2007) provided a generation model based on data mining of historic intrusion alerts.

According to Hsu and Lin (2008), Attack Graphs are faced with a combination explosion in terms of their complexity and are therefore always applied to smaller network systems whereas consideration of large networks is subjective to some system modifications (Noel and Jajodia, 2004). Noel and Jajodia (2009a) used a model checking strategy to list the attack chains to link the privilege of the initial attacker to the final objective of the assault. This method also improves exponentially as the network size rises due to a large number of attack states being enumerated. The assumption of monotonicity in the logic used during the Attack Graph generation, however, reduced complexity to polynomial. In consideration of a quadratic number of hosts, the complexity of such graphs was reduced.

Noel and Jajodia (2009b) grouped networks into single domains with no restriction of connectivity between hosts and tight security protection rules were applied to such domains. This approach aimed to reduce the Attack Graph's complexity. In this job, the topology proposed decreased the complexity of single domain consideration to linear. Depending on the number of protected domains, the number draws to a quadratic (as the number represents the domain number, not the host). However, the graphs generated with this strategy ranged from hundreds to tens of thousands of hosts that were produced within minutes without visualizations. Hong et al. (2013) provided a scalable model of attack representation using a method of logic reduction. The work proposed a method of simplifying the attack tree based on the tree's logical expression. It showed an equivalent safety assessment before and after reducing the Attack Graph's logic expressions. The methods of logic reduction were used to automate the building as well as to decrease the size of the attack trees. The complexity of the attack trees generated was analyzed and simulation was performed using different network topologies to evaluate the performance of the logic reduction techniques. The complexity analysis conducted in the job showed that the Attack Graph size was lower than the complete Attack Graphs after the logic reduction. It also described the trade-off between the moment the tree was built and the use of memory.

Lee et al. (2009) proposed a mechanism for Attack Graph management using a divide and conquer approach. A large Attack Graph was converted into multiple sub graphs to enhance the efficiency of the Attack Graphs risk analyzer. The outcome showed that when k time complexity algorithms are used with an Attack Graph with n vertices, a division with c overhead vertices would decrease the workload from n^ktor$(n + c)^k$. The workload decrease will allow the risk assessment of the big Attack Graph to become more scalable and practical. The approach to divide and conquer presented in this work did not require any adaptation of methods of risk analysis. Risk units, also known as light graphs, have been used to reduce the analyzers' workloads.

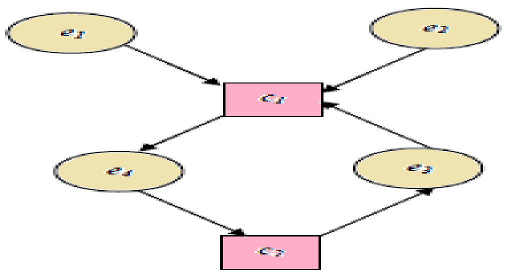
Ma et al. (2010) provided a scalable, two-way search approach for Attack Graphs generation. The target network used in this job was based on four levels: network service, host system, security system, and host accessibility. In this research, a technology that can automatically obtain the parameters of the accessibility of the host was provided. This technology helped to automatically model a large-scale network and also reduced the spatial complexity of the proposed algorithm in this work. To aggregate and generate the host Attack Graph whose number of nodes and edges increase linearly with the number of hosts in the network, vulnerabilities, and attacks were linked to specific hosts according to the predefined rules of the network system.

Several vulnerability identification and measurement techniques are available, such as the Vulnerability Rating and Scoring System (VRSS) and the Common Vulnerability Scoring System (CVSS) (Scarf one and Mell 2009). These systems of scoring are based on known vulnerability experiences. For example, before successfully exploiting the vulnerability, the level of privileges an attacker must possess the conditions that are beyond the control of the attacker that must exist to exploit the vulnerability or a user's requirements other than the attacker to successfully compromise the

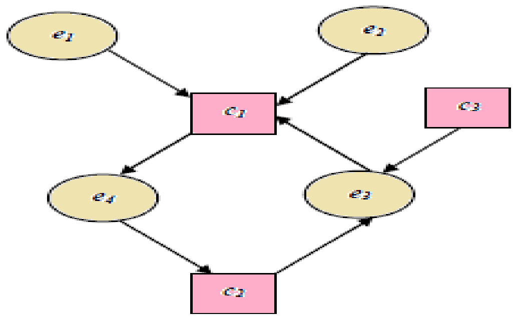
vulnerable services. The vulnerabilities will then be assigned numeric scores. While these approaches focus on individual vulnerabilities, as each vulnerability may be scored low, a network security expert may be misled.
MulVAL has been developed based on multi-host, multi-stage vulnerability analysis. It's a Kansas State University open-source project. Data log has been used to describe networks and their safety rules and conditions. The rule files are scanned by the execution engine of Prolog (Ou (2005); Ou et al (2006); Ou et al (2005)).

III. CYCLES IN ATTACK GRAPHS

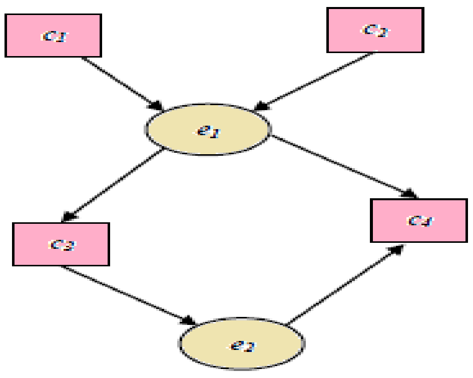
Attack Graph is a cycle graph that is directed. The effect of cycles on Attack Graphs is a major complication in Attack graph models. There are different types of cycles in Attack Graphs which could naturally exist. These cycles create different problems. Figure 1 presents various cycle cases that have been considered in this paper. The interaction formed the cycles in Figure 1 (Ou et al, 2006).
The logic programming language data logs are used by MulVAL Attack Graph to describe the networks and their safety rules and conditions. The conditional nodes (c_1, c_2,..c_n) are OR-decomposed nodes, while the AND-decomposed nodes are the exploit nodes (e_1, e_2, e_3., e_n). Some cycles can be removed from the Attack Graph completely while others can't. Removal of cycles depends on whether attackers can ever reach any of the exploits or conditions inside the cycle. This implies that a cycle can be removed if an attacker cannot reach any of its exploits or conditions otherwise such a cycle is irremovable from the Attack Graph.



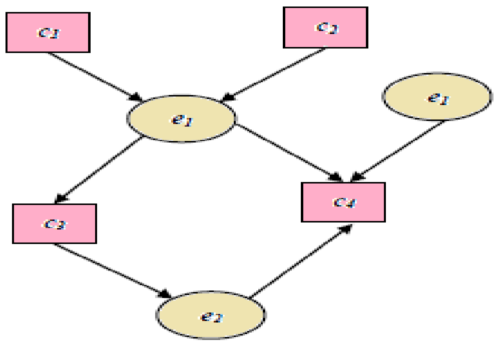
(a)



(b)



(c)



(d)

Figure. 1 Different case of Attack Graph Cycles considered

The cycle can be reached by a conditional node c_1 in Figure 1(a). One of the e_1, e_2, e_3 exploit nodes can achieve this node. The operating node e_4 relies on c_1, c_2 on e_4 and e_3 on the conditional node c_2. This means that e_4 and c_2 can be reached as they rely on c_1 if the situation node c_1 is satisfied. If it is possible to reach c_2, then e_3 can be successfully exploited. This Attack Graph cycle example cannot be removed as it is possible to reach all attack nodes and privileges during the attack. Figure 1(b) only shows a cycle similar to (a) that the conditional node c_3 now depends on the attack node e_3. It is possible to reach the cycle via a conditional node c_1. c_1 is an OR-node that can be reached through any of the e_1, e_2, e_3.
Attack node e_3 requires both c_2 and c_3 nodes before being successfully exploited. It is also possible to reach the two attack nodes in the cycle, as both depend on c_1. The Attack Graph cannot remove this type of cycle. Figure 1(c) shows a different Attack Graph cycle case. It is possible to reach the cycle through the attack node e_1. e_1 is an AND-node that requires c_1, c_2, and c_4 predecessors to be satisfied before it can be used successfully. c_3 and e_2 depend on e_1 and c_4 attack node as well as e_2. It can be seen that both e_1 and c4 nodes depend on each other and therefore cannot be reached during the attack. An Attack Graph can remove this type of cycle. Figure 1(d) shows a similar cycle of (c) with the attack node e_3 required to satisfy the conditional node c_4. The attack node e_1 needs

International Innovative Research Journal of Engineering and Technology
ISSN: 2456-1983 Vol: 5, Issue No: 1, September 2019

the satisfaction of all nodes c_1, c_2, and c_4 before exploiting the attack. Before it can be achieved, the conditional node c_4 requires e_2 or e_3 as a precondition. Nodes e_2 and c_3 all rely on the exploit node e_1, but c_4 is an OR node that can either be affected without e_2 based on the external exploit node e_1. This implies that by using the external node e_3, the relationship that exists between nodes c_4 and e_1 can be broken. Therefore, during an assault, all the privilege and attack nodes can be reached. The Attack Graph cannot remove this cycle.

IV. PROPOSED APPROACH

This paper's proposed approach leverages graph theories. The approach can be divided into three parts: Attack Graph generation, Attack Graph identifying and removing cycles in the Attack Graph and Attack Graph determining valid attack paths. The detailed flowchart diagram of the proposed approach, i.e. the process of generating the Attack Graph and scaling the Attack Graph generated, is presented in Figure 2, the details are further described in the following procedure.

i. OAUNET was chosen as an issue domain (i.e. network environment).
ii. Using the Nessus scanner tool, the mapping of network connections and domain knowledge of vulnerability information were identified. The mapping of the network connection involved information that was available throughout the target network hosts. This includes topology or connectivity information (unique host identifiers such as host IP and hostname), services running on the hosts, and vulnerabilities in operating systems, software, and services that have security flaws in the network hosts. Besides, the vulnerability information domain knowledge was identified using NVD. This shows the dependency or relationship between the various vulnerabilities in the target network.
iii. The report of Nessus vulnerability scanning has been exported as .nessus file. In the Nessus scanning result (.nessus), the MulVAL takes as input, which is then translated into MulVALdatalogs.
iv. The Attack Graph was created using the MulVAL framework, in this paper the details were handled offline.
v. All cycles were identified in the generated Attack Graph and it was decided whether or not to remove them using the proposed cycle handling algorithm in Figure 3 (Cycle Detection and Handling).
vi. In Figure 4 (Attack Graph Scaling) the Attack Graph was scaled using the proposed Valid Path Algorithm.

4.1 Proposed Cycle Handling Algorithm

An Attack graph is a directed cyclic graph (DCG). It contains some set of strongly connected components where there are strongly connected subsets of the vertices (exploits and conditions). Johnson (1975) introduced an algorithm capable of detecting all possible cycles in a directed graph. The detection of cycles of Attack Graphs will improve this algorithm. The improved algorithm shown in Figure 3 demonstrates how cycles can be treated in Attack Graphs. This algorithm uses the directed cyclic Attack Graph as input and treats the cycles in the Attack Graph as heavily connected elements. Each cycle is subject to exploiting reach ability in the set of strongly connected components found in the attachment graph. This is important to determine whether or not the cycle is removable.

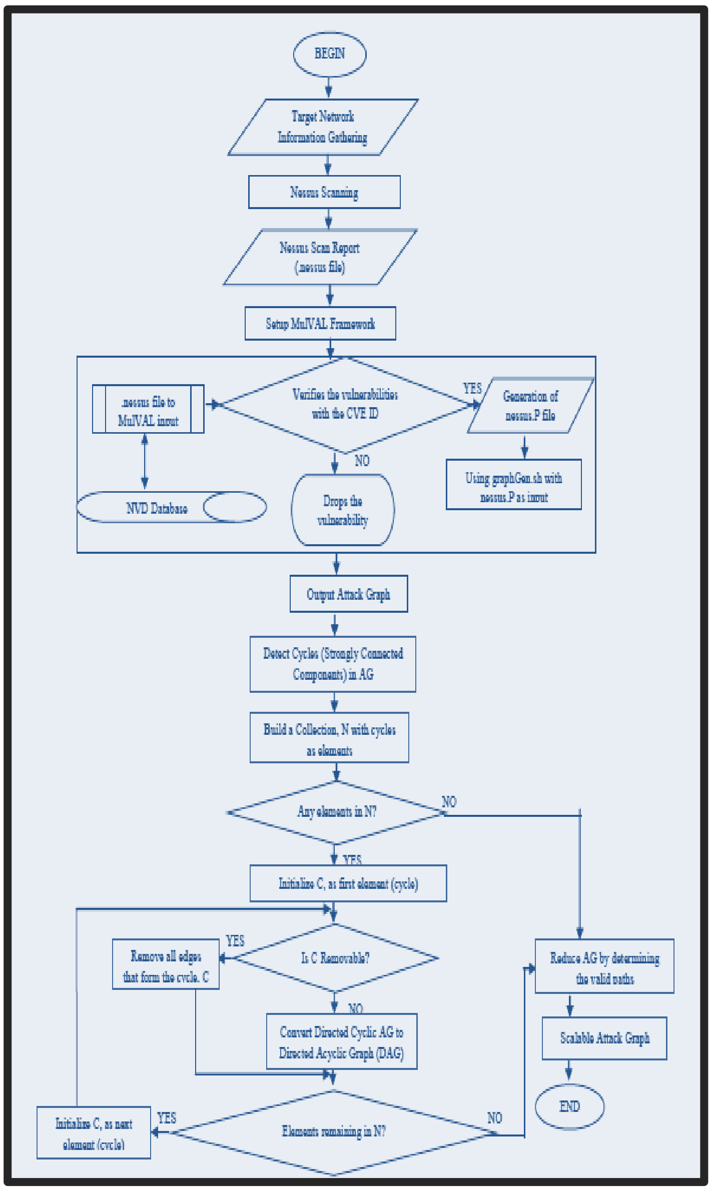
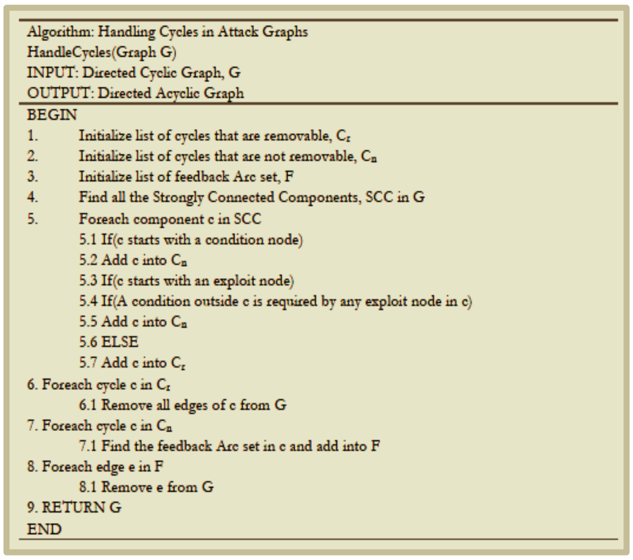

BEGIN
Target Network Information Gathering
Nessus Scanning
Nessus Scan Report (.nessus file)
Setup MulVAL Framework
.nessus file to MulVAL input
Verifies the vulnerabilities with the CVE ID
YES
Generation of nessus.P file
NO
NVD Database
Drops the vulnerability
Using graphGen.sh with nessus.P as input
Output Attack Graph
Detect Cycles (Strongly Connected Components) in AG
Build a Collection, N with cycles as elements
Any elements in N?
NO
YES
Initialize C, as first element (cycle)
YES
Remove all edges that form the cycle. C
Is C Removable?
NO
Convert Directed Cyclic AG to Directed Acyclic Graph (DAG)
Reduce AG by determining the valid paths
Scalable Attack Graph
END
Initialize C, as next element (cycle)
YES
Elements remaining in N?
NO

Figure 2. Flow chart

However, those cycles that cannot be removed within the Attack Graphs are subjected to a Feedback arc test, where edge sets can be removed to convert the directed cyclic graph to a directed acyclic graph.

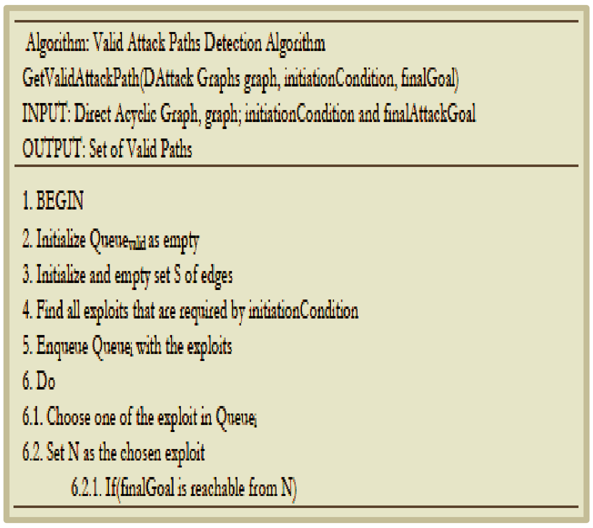
4.2 Proposed Valid Attack Path Detection Algorithm

The Attack Graphs generated by previous approaches do not scale as node size increases in a network system. A node may have hundreds of vulnerabilities, which may also constitute the Attack Graph's exponential growth. Such Attack Graphs are difficult to visually interpret in a business network for network system security risk management. To address these scalability issues of Attack Graphs generated, this research proposed in Figure 4 a forward search-based algorithm that identifies the Attack Graphs' valid attack paths. The valid Attack Graph paths are easier and faster to understand than the Attack Graphs generated.



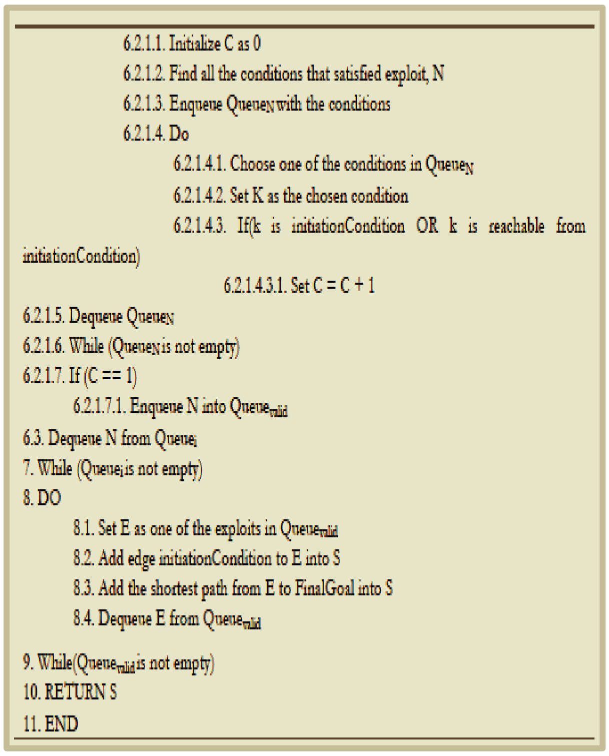
```
Algorithm: Handling Cycles in Attack Graphs
HandleCycles(Graph G)
INPUT: Directed Cyclic Graph, G
OUTPUT: Directed Acyclic Graph
BEGIN
1.      Initialize list of cycles that are removable, $C_r$
2.      Initialize list of cycles that are not removable, $C_n$
3.      Initialize list of feedback Arc set, F
4.      Find all the Strongly Connected Components, SCC in G
5.      Foreach component c in SCC
        5.1 If(c starts with a condition node)
        5.2 Add c into $C_n$
        5.3 If(c starts with an exploit node)
        5.4 If(A condition outside c is required by any exploit node in c)
        5.5 Add c into $C_n$
        5.6 ELSE
        5.7 Add c into $C_r$
6. Foreach cycle c in $C_r$
        6.1 Remove all edges of c from G
7. Foreach cycle c in $C_n$
        7.1 Find the feedback Arc set in c and add into F
8. Foreach edge e in F
        8.1 Remove e from G
9. RETURN G
END
```

Figure 3. Algorithm - Handling cycles in Attack Graphs



```
Algorithm: Valid Attack Paths Detection Algorithm
GetValidAttackPath(DAttack Graphs graph, initiationCondition, finalGoal)
INPUT: Direct Acyclic Graph, graph; initiationCondition and finalAttackGoal
OUTPUT: Set of Valid Paths

1. BEGIN
2. Initialize $Queue_{valid}$ as empty
3. Initialize and empty set S of edges
4. Find all exploits that are required by initiationCondition
5. Enqueue $Queue_i$ with the exploits
6. Do
6.1. Choose one of the exploit in $Queue_i$
6.2. Set N as the chosen exploit
        6.2.1. If(finalGoal is reachable from N)
```



```
            6.2.1.1. Initialize C as 0
            6.2.1.2. Find all the conditions that satisfied exploit, N
            6.2.1.3. Enqueue $Queue_N$ with the conditions
            6.2.1.4. Do
                6.2.1.4.1. Choose one of the conditions in $Queue_N$
                6.2.1.4.2. Set K as the chosen condition
                6.2.1.4.3. If(k is initiationCondition OR k is reachable from initiationCondition)
                    6.2.1.4.3.1. Set C = C + 1
6.2.1.5. Dequeue $Queue_N$
6.2.1.6. While ($Queue_N$ is not empty)
6.2.1.7. If (C == 1)
        6.2.1.7.1. Enqueue N into $Queue_{valid}$
6.3. Dequeue N from $Queue_i$
7. While ($Queue_i$ is not empty)
8. DO
        8.1. Set E as one of the exploits in $Queue_{valid}$
        8.2. Add edge initiationCondition to E into S
        8.3. Add the shortest path from E to FinalGoal into S
        8.4. Dequeue E from $Queue_{valid}$

9. While($Queue_{valid}$ is not empty)
10. RETURN S
11. END
```

Figure 4. Valid Attack Path Detection Algorithm.0

V. RESULT & DISCUSSION

5.1 Attack Graph Generation

Figure 5 shows the Attack Graph with 103 nodes being generated. For better visualization, it was rendered with numeric values assigned to each node. This MulVAL Attack Graph's AND nodes are shaped as ellipses, while the diamond-shaped OR nodes and vulnerability nodes are inboxes. The leaf nodes are the configurations that usually have no ancestor on each host of the network system. Based on Figure 5's critical observation, it was noted that the Attack Graph generated is quite large and too complex to understand.

5.2 Directed Acyclic Graph Generation

JAVA programming language used Netbeans IDE 8.0.2 running on Java Development Kit (JDK) 1.8.0 Update 25 to implement the algorithm in Figure 3. The algorithm has been tested with the Attack Graph generated in Figure 5. In this Attack Graph, there are 153 cycles. All the cycles found in this Attack Graph are listed in the Appendix, but all these cycles are not removable, unfortunately. In the feedback arc set, the Attack Graph included 7 edges. This set of edges includes edges that could be removed in the Attack Graph to create a directional acyclic graph. Table 1 shows the list of the 7 edges. The removal of these edges produced the Directed Acyclic Graph required as shown in Figure 6.

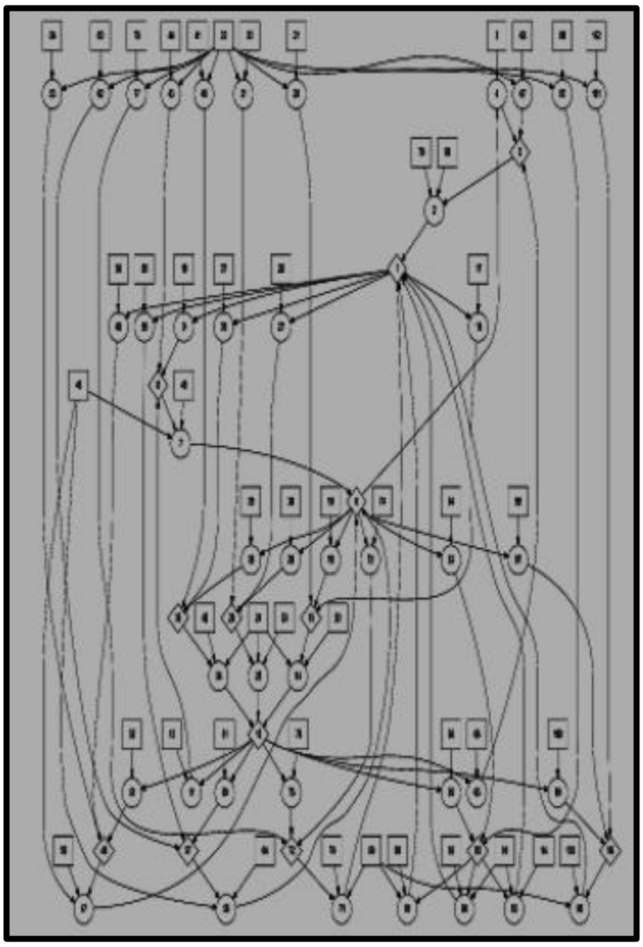


Figure 5. The Generated MulVAL Attack Graph (Node numbering)

Table 1. The Feedback arc Set of the generated Attack Graph

S/N	Edges
1	**96->95**
2	**25->13**
3	**83->82**
4	**34->13**
5	**2->1**
6	**14->13**
7	**71->1**

5.3 Scalable Attack Graph Generation

The Figure 4 algorithm takes as input the Directed Acyclic Graph, the condition of the initial attacker and the goal of the final attacker. This algorithm's output is a collection of valid routes that created the Attack Graph required to manage security risk. The implementation of this algorithm was tested using Figure 6 presented with the Directed Acyclic Attack Graph.

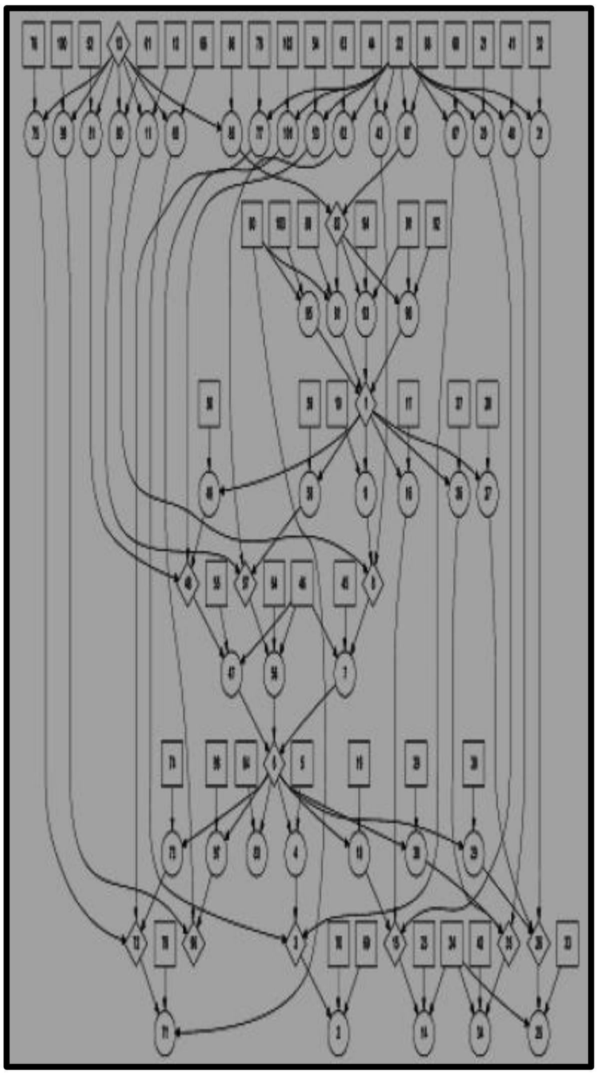


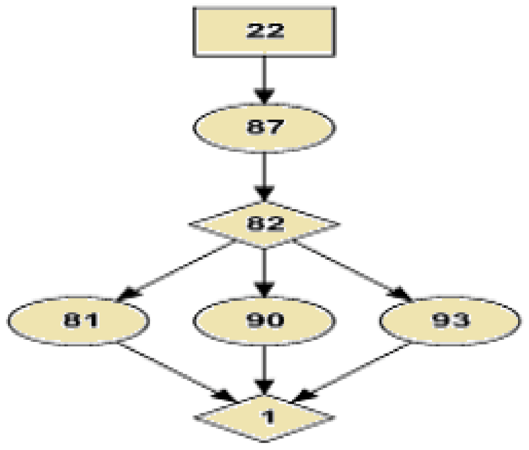
Figure 6. Generated Directed Acyclic Attack Graph

The initial condition of the attacker is node 22, which presents the attacker locating the Internet to exploit the remote network. Nodes 1, 6 and 13 are the ultimate goal of the attacker (to execute some exploit codes on each target host). This work assumes that an attacker independently exploits each of the final goals. Figure 7 shows the results of this algorithm being implemented with Figure 7(a) and Figure 7(b) showing the valid paths using nodes 1 and 6 respectively as the final targets for the attack. There is however no valid path from the initial condition goal of the attacker (node 22) and node 13 attack goal.
The final Attack Graph generated by merging the subgraphs in Figure 7(a) and (b) is presented in Figure 8. In terms of the graph's overall size, the final output is more scalable and easier to interpret or understand than the one generated in Figure 5. Furthermore, the MulVAL Attack Graph in Figure 5 has a total logical size of 2.29 MB, with a total size of 1.86 MB the directed acyclic Attack Graph is shown in Figure 6 is lighter. The total

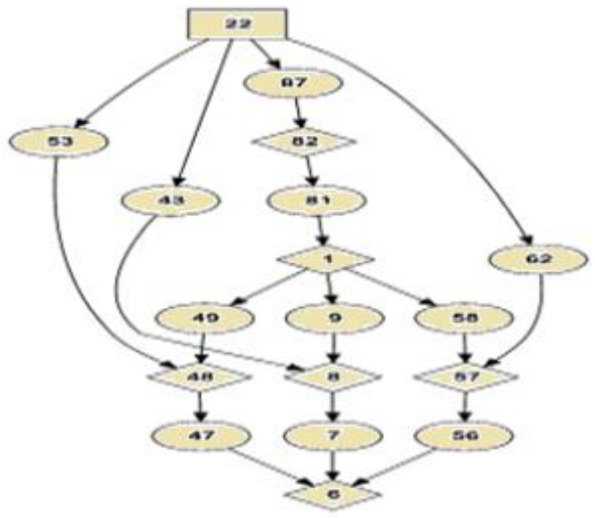
logical size of the enhanced Attack Graph generated from this study as shown in Figure 8 is 0.97 MB.

VI. CONCLUSION

This paper presented a graph-theoretical approach to address Attack Graphs' scalability issues. For detecting and managing the cycles that are always present in Attack Graphs, an algorithm was formulated. These cycles can be removed or not, depending on whether an attacker can reach the exploit node in the cycle. This paper also presents a valid algorithm for attack path detection that can be used to determine an Attack Graph's most critical and valid path. The proposed strategy will improve network system safety evaluation that is visually dependent on Attack Graphs.
This reduces the problem of scalability of such an Attack Graph that is growing exponentially



(a)



(b)

Figure 7. Valid Attack Paths from the initial attacker's condition

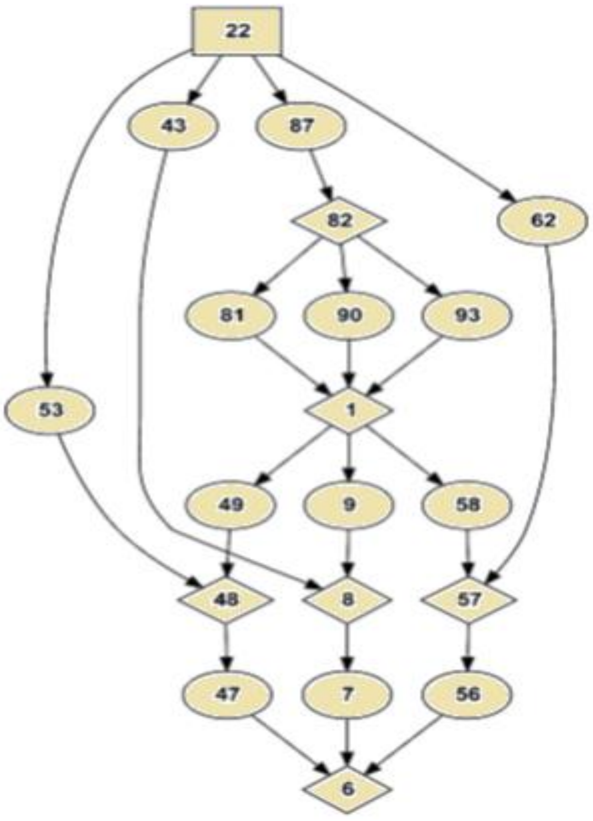


Figure 8. Enhanced Attack Graph

Increasing network host size and vulnerability. It will thus enable the network managers to make quick decisions during such vulnerability assessment.

REFERENCES

[1] Ammann P., Wijesekera D., and Kaushik S., (2002) "scalable, graph-based network vulnerability analysis", In Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS), ACM Press, Washington, USA, November 18-22, 217-224.

[2] Ammann P., Pamula J., Ritchey R., and Street J., (2005) "A host-based approach to network attack chaining analysis", In Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC '05), IEEE Computer Society, Tucson, Arizona, December 5-9, 10

[3] Bhattacharya S., Malhotra S., and Ghsoh S., (2008) "A scalable representation towards attack graph generation," 2008 1st International Conference on Information Technology, DOI:10.1109/INFTECH.2008.4621611. 1–4.

[4] Dacier M., Deswarte Y., and Kaaniche M., (1996). "Quantitative assessment of operational security: Models and tools," 96493, Tech. Rep., [Online]. Available: http://citeseer.ist.psu.edu/366225.html

[5] Eades, P., Lin, X., & Smyth, W. F. (1993). A Fast and Effective Heuristic for the Feedback Arc Set Problem. Information Processing Letters, 47(6), 319-323.

[6] Hong, J. B., Kim, D. S., & Takaoka, T. (2013). Scalable Attack Representation Model Using Logic Reduction Techniques. 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2013,. 404-411.

[7] Hsu, L. H., & Lin, C. K. (2008). Graph theory and interconnection networks. CRC press.

[8] Jajodia S., Noel S., and O‟Berry B., (2005), "Topological analysis of network attack vulnerability", in Managing Cyber Threats: Issues, Approaches and Challenges, V. Kumar, J. Srivastava, A. Lazarevic (eds.), Springer, 2005.

[9] Johnson, D. B. (1975). Finding All the Elementary Circuits of a Directed Graph. SIAM Journal on Computing, 4(1), 77-84.

[10] Lee, J., Lee, H., & In, H. P. (2009). Scalable Attack Graph for Risk Assessment. In International Conference on Information Networking, 2009 (ICOIN 2009). 1-5

[11] Ma, J., Wang, Y., Sun, J., & Hu, X. (2010). A Scalable, Bidirectional-Based Search Strategy to Generate Attack Graphs. In IEEE 10th International Conference in Computer and Information Technology (CIT), 2010, 2976-2981.

[12] Man D., Zhang B., Yang W., Jin W., and Yang Y., (2008) "A method for global Attack Graph generation," IEEE International Conference on Networking, Sensing and Control, 2008. ICNSC 2008, 236–241.

[13] Noel S., and Jajodia S., (2004)"Managing Attack Graph Complexity through Visual Hierarchical Aggregation", In Proceedings of the ACM workshop on Visualization and data mining for computer security (VizSEC/DMSEC „04), ACM, VA, USA, October 29, 2004, 109–118

[14] Noel, S., &Jajodia, S. (2009a). Proactive Intrusion Prevention and Response via Attack Graphs. in Practical Intrusion Analysis: Prevention and Detection for the Twenty-First Century, R. Trost (ed.), Addison-Wesley Professional, 2009

[15] Noel, S., &Jajodia, S. (2009b). "Advanced Vulnerability Analysis and Intrusion Detection through Predictive Attack Graphs," Critical Issues in Command, Control, Communications, Computers, Intelligence (C4I), Armed Forces Communications and Electronics Association (AFCEA) Solutions Series, Lansdowne, Virginia, May 2009.

[16] Ortalo R., Deswarte Y., and Kaaniche M. (1999), "Experimenting with quantitative evaluation tools for monitoring operational security," IEEE Trans. Software Eng, 25(5), 633– 650,

[17] Ou, X., (2005). A Logic-Programming Approach to Network Security Analysis. PhD thesis, Princeton University, 2005.

[18] Ou, X., Boyer, W. F., & McQueen, M. A. (2006). A Scalable Approach to Attack Graph Generation. In Proceedings of the 13th ACM Conference on Computer and Communications Security.336-345.

[19] Ou, X., Govindavajhala, S., &Appel, A. W. (2005). MulVAL: A Logic-based Network Security Analyzer. In 14th USENIX Security Symposium, 2005.

[20] Phillips C. A. and Swiler L. P.(1998). "A graph-based system for network-vulnerability analysis,". In NSPW '98: Proceedings of the 1998 workshop on New security paradigms, 71–79.

[21] Scarfone K. and Mell P.(2009) "An Analysis of CVSS Version 2 Vulnerability Scoring," in Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ser.ESEM "09. Washington, DC, USA: IEEE Computer Society, 516–525.

[22] Sheyner O., Haines J. W., Jha S., Lippmann R., and Wing J. M., (2002) "Automated generation and analysis of Attack Graphs," in IEEE Symposium on Security and Privacy, 273–284.

[23] Sheyner O. M.,(2004) "Scenario graphs and Attack Graphs," Ph.D. dissertation, Carnegie Mellon University, 2004.

[24] Swiler L., Phillips C., Ellis D., and Chakerian S.,(2001) "Computer Attack Graph generation tool," in Proceedings of DARPA Information Survivability Conference and Exposition II (DISCEX'01), 2001. DOI: 10.1109/DISCEX.2001.932182

[25] Tang Li Z., Lei J.,Wang L., and Li D.,(2007) "A data mining approach to generating network attack graph for intrusion prediction," Fourth International Conference on Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007., 4, 307–311.