American International Journal of Computer Science and Information Technology
Volume.8, Number 3;July-September, 2023;
ISSN: 2837-388X| Impact Factor:
https://zapjournals.com/Journals/index.php/aijcsit
Published By: Zendo Academic Publishing

A COMPREHENSIVE STUDY ON SCALABLE ATTACK GRAPH GENERATION FOR NETWORK SECURITY

Engr. Olufemi Olatunji Adeboye[1] and Dr. Jean-Claude Kouakou Gbocho[2]

Article Info

Keywords: Computer, Networks, Network Security Vulnerabilities, Intrusion Prevention, Malicious Intrusion

Abstract

In the contemporary landscape, computer networks have assumed pivotal roles within the global economy and national infrastructures. Their indispensability is palpable across diverse sectors encompassing economics, finance, and commerce. Network systems function as intricate amalgamations of software, services, and configurations, collectively orchestrated to facilitate seamless data communication. However, this complexity renders them inherently susceptible to a burgeoning array of vulnerabilities. Consequently, fortifying network security has emerged as a paramount challenge, necessitating comprehensive evaluation to shield against potential malicious intrusions. Intrusion prevention stands out as a potent strategy in fortifying network security, entailing the identification and mitigation of vulnerabilities or attack vectors within the network architecture. This proactive approach initiates with the detection of potential threats, equipping stakeholders with the critical understanding of attacker methodologies and potential exploit vectors. This knowledge forms the bedrock for preemptive security measures, ensuring the integrity and resilience of network systems prior to the implementation of network hardening protocols

1. Introduction

Computer Networks play important roles in today‟s economy and national infrastructures. The dependencies on them are increasing in various fields of economic, financial, business etc. Network systems are availed with different software, services and configuration running dependently together for the purpose of data communication. These systems are vulnerable and the vulnerabilities are increasing every year. Therefore, network security has become one of the main challenges these days and has to be evaluated in order to protect the network against any form of malicious intrusion. Intrusion prevention is one of the effective approaches for improving network security and it involves eliminating cause of attacks or vulnerabilities in the network. Intrusion prevention starts with the detection of possible attacks in the networks or having a knowledge about

[1] Obafemi Awolowo University, Ile-Ife, Nigeria.
[2] Laboratoire LARIT, Cocody Danga, Abidjan, Côte d'Ivoire.

how attackers can exploit the vulnerability of the network to breach the security and obtain the attack goal before network hardening.

An attacker can exploit multiple vulnerabilities in a network before achieving a particular goal e.g getting root privilege into a server. Such attacks are called multi-step attacks. Attack Graph is a powerful tool that can provide information about the relationship that exist among various vulnerabilities that can be exploited by the attacker and the privileges gained by the attacker as a result of exploiting these vulnerabilities. Attack Graph shows the possible sequence or paths of malicious actions that an attacker can follow to intrude into the network and gain certain privileges. These vulnerabilities could be as a result of inappropriate configuration setting in the network system or existence of a specific version of a software product.

Attack Graph considers the number of vulnerabilities on the target network, the conditions that define the reachability among the vulnerable software instances and the level of detail in vulnerability modeling. All these influence the size of the Attack Graph. This means, the bigger the network size, the bigger the size of the Attack Graph. The more the number of vulnerabilities on the target network, the bigger the size of the Attack Graph. This implies that as the hosts in a network grow in size, it becomes more difficult to evaluate and automate their vulnerability to attack, so the Attack Graph become very large and complex. The scalability problem of Attack Graph is therefore necessary and required in network systems for the purpose of network hardening and network security risk management.

This study aimed at developing a model that increases the scalability of Attack Graph for the purpose of network security analysis. This study was motivated by the use of Attack Graphs used by network administrators to acquire knowledge about how attackers can combine multiple vulnerabilities in network systems in order to breach certain security goals. This process is quite challenging as the Attack Graphs are not easy to comprehend or interpret when the size of the network grows or when the vulnerabilities in the target network system grow in number.Hence, there is a need to provide a more scalable Attack Graph which captures only the required problem area which the network administrator leverage on during the vulnerability analysis. As a result of this, the study looks into the cycles that are associated with Attack Graphs and determine whether those cycles can be removed or not. It also determine the most critical valid paths in Attack Graph as required by network administrators during network security analysis.The rest of this paper is arranged as follows: Section 2 and 3 discusses the related works while Section 4 describes the proposed approach, Section 5 describes the expected result and the summary and conclusions are discussed in Section 6.

2. Related Works

Generation of Attack Graph was first done using the *red team* approach, this was prone to errors and very tedious as it was based on manual effort which was not suitable for moderate size network. Different approaches have been proposed to automatically generate Attack Graphs.The concept of Attack Graph was proposed by Phillips and Swiller (1998) and a tool was presented by Swiller *et al* (2001) to generate Attack Graph. Attack templates were used to represent generic steps in known attacks in their model. The concept of privilege graph was proposed by Dacier *et al* (1996), after which the use of the privilege graph in network security was illustrated by Ortalo *et al* (1999). The Attack Graph however, could not be computed as they turned out to be too large even with only 13 vulnerabilities.

In Sheyner *et al.* (2002) and Sheyner (2004) a model checker, NuSMV was used to compute multi-stage, multihost Attack Graph. Ammann *et al.* (2002) used the assumption of monotonicity to address the problem of scalability associated with Attack Graph and was able to reduce the computational cost to polynomial successfully. Jajodia *et al*. (2005) implemented an integrated, topological approach to vulnerability analysis using the algorithm presented by Ammann *et al.* (2002). This approach was called Topological Vulnerability Analysis (TVA).

The different parts of the exploit-dependent Attack Graph generated by TVA in Jajodia *et al*. (2005) were collapsed to make visual understanding more interactive in the work of Noel and Jajodia (2004). Ammann *et al*. (2005) computed the suboptimal attack path among every pair of hosts in a network through an algorithm. This work could find the maximum privilege that can be gained on each host as the attacker exploits the vulnerabilities of the network.A breadth-first generation algorithm was proposed by Man *et al.* (2008) by adding the attack step and success probability in order to limit the scale of the graphs. Bhattacharya *et al*. (2008) proposed a generic attack path detection algorithm and showed that the attack paths are scalable. A generation model based on data mining of historical intrusion alerts was presented by Tang *et al.* (2007).

According to Hsu and Lin (2008), Attack Graphs face a combinational explosion with respect to their complexity and thus, they are always applied to smaller network systems while the consideration for large networks is subjective to some system modifications (Noel and Jajodia*,* 2004).Noel and Jajodia (2009a) used a model checking approach to enumerate the attack chains for the purpose of linking initial attacker‟s privilege to the final attack goal. This approach also grows exponentially as the size of the network increases due to the enumeration of large number of attack states. However, the assumption of monotonicity in the logic used during the generation of Attack Graph reduced the complexity down to polynomial. The complexity of such graphs were reduced while considering a quadratic number of hosts.

Noel and Jajodia (2009b) grouped networks into single domain with no restriction in the connectivity among the hosts and such domain had tight security protection rules applied. This approach was aimed at reducing the complexity of the Attack Graph. The topology proposed in this work reduced the complexity to linear considering single domain. The number draws to a quadratic depending on the number of the protected domains (as the number represents the domain number and not host). The graphs generated with this approach ranged from ones of hundreds to tens of thousands of hosts were generated within minutes however, with no visualizations.Hong *et al.,* (2013) presented a scalable attack representation model using a logic reduction technique. The work proposed an attack tree simplification method based on logical expression of the attack tree. It demonstrated an equivalent security assessment before and after the reduction of the logic expressions of the Attack Graph. The logic reduction techniques were used to automate the construction and also reduce the size of the attack trees. The complexity of the attack trees generated were analysed and a simulation was done to evaluate the performance of the logic reduction techniques using various network topologies. The complexity analysis conducted in the work showed that the size of the Attack Graph after the logic reduction was smaller than the full Attack Graphs. It also presented the trade-off between the time of construction of the attack tree and the memory usage.

Lee *et al.* (2009) proposed Attack Graph management mechanism using a divide and conquer approach. A large Attack Graph was converted to multiple sub-graphs for the purpose of enhancing the efficiency of risk analyzer used with Attack Graphs. The result showed that when **k** order of time complexity algorithms are used with an Attack Graph with **n** vertices, a division with **c** overhead vertices would reduce the workloads from $n^k$to $r(n + c)^k$. The workload reduction will allow risk assessment on large Attack Graph to become more scalable and practical. The divide and conquer approach presented in this work did not require any adaptation of risk analysis methods. Risk Units, also known as light-weighted graphs were used to reduce the workloads of the analyzers.

Ma *et al.* (2010) presented a scalable, bidirectional-based search strategy to generate Attack Graphs. The target network used in this work, was modeled in four levels: network service, host system, security system and accessibility of host. A technology that can acquire the parameters of the host‟s accessibility automatically was presented in this study. This technology helped in modeling a large-scale network automatically and also reduced the space complexity of the algorithm proposed in this work. Vulnerabilities and attacks were linked to specific hosts according to the predefined rules of the network system in order to aggregate and generate the host Attack Graph whose number of nodes and edges increase linearly with the number of hosts in the network. This approach resolved the shortcomings of the state enumeration Attack Graphs, in which as the number of host in the network increases, the Attack Graph grows exponentially. In addition, the forward and reverse search threads were performed at the same time in order to reduce the depth of search threads. The work was also based on the assumption of monotonicity to generate the Attack Graphs using the bidirectional-based search strategy.

There are several techniques for identifying and measuring individual vulnerabilities, such as the Vulnerability Rating and Scoring System (VRSS) and the Common Vulnerability Scoring System (CVSS) (Scarfone and Mell, 2009). These scoring systems are based on known experiences about vulnerabilities. For example, the level of privileges an attacker must possess before exploiting the vulnerability successfully, the conditions that are beyond the attacker‟s control that must exist in order to exploit the vulnerability or the requirements for a user, other than the attacker, to compromise the vulnerable services successfully. Numeric scores are then assigned to the vulnerabilities. While these approaches focus on individual vulnerabilities, a network security expert could be misled as each individual vulnerability may be scored low. This is because, there are interactions among those vulnerabilities and that, attackers combine sequences of related vulnerabilities to evade network security measures.

MulVAL was developed based on Multi-host, Multi-stage Vulnerability Analysis. It is an open source project in Kansas State University. Logic programming language Datalog was used to describe the networks and their security rules and conditions.The rule files are parsed by the Prolog execution engine (Ou (2005); Ou *et al* (2006); Ou *et al* (2005)).

3. Cycles In Attack Graphs

Attack graph is a directed cycle graph. A major complication in Attack Graphs models lies in the effects of cycles on Attack Graphs. There exists different types of cycles that could naturally exist in Attack Graphs. These cycles create different difficulties. Figure 1 presents different cases of cycles that were considered in this paper. The cycles in Figure 1 are formed through the interaction (Ou *et al,* 2006).

MulVAL Attack Graph use the logic programming language datalogs to describe the networks and their security rules and conditions. The conditional nodes ($c_1, c_2, \ldots c_n$) are OR-decomposed nodes while the exploit nodes ($e_1, e_2, e_3 \ldots, e_n$) are the AND-decomposed nodes. Some cycles can be removed completely from the Attack Graph while some cannot. Removal of cycles depends on whether any of the exploits or conditions inside the cycle can ever be reached by attackers or not.This implies that, a cycle can be removed if none of its exploits or conditions can be reached by an attacker else such cycle is irremovable from the Attack Graph.

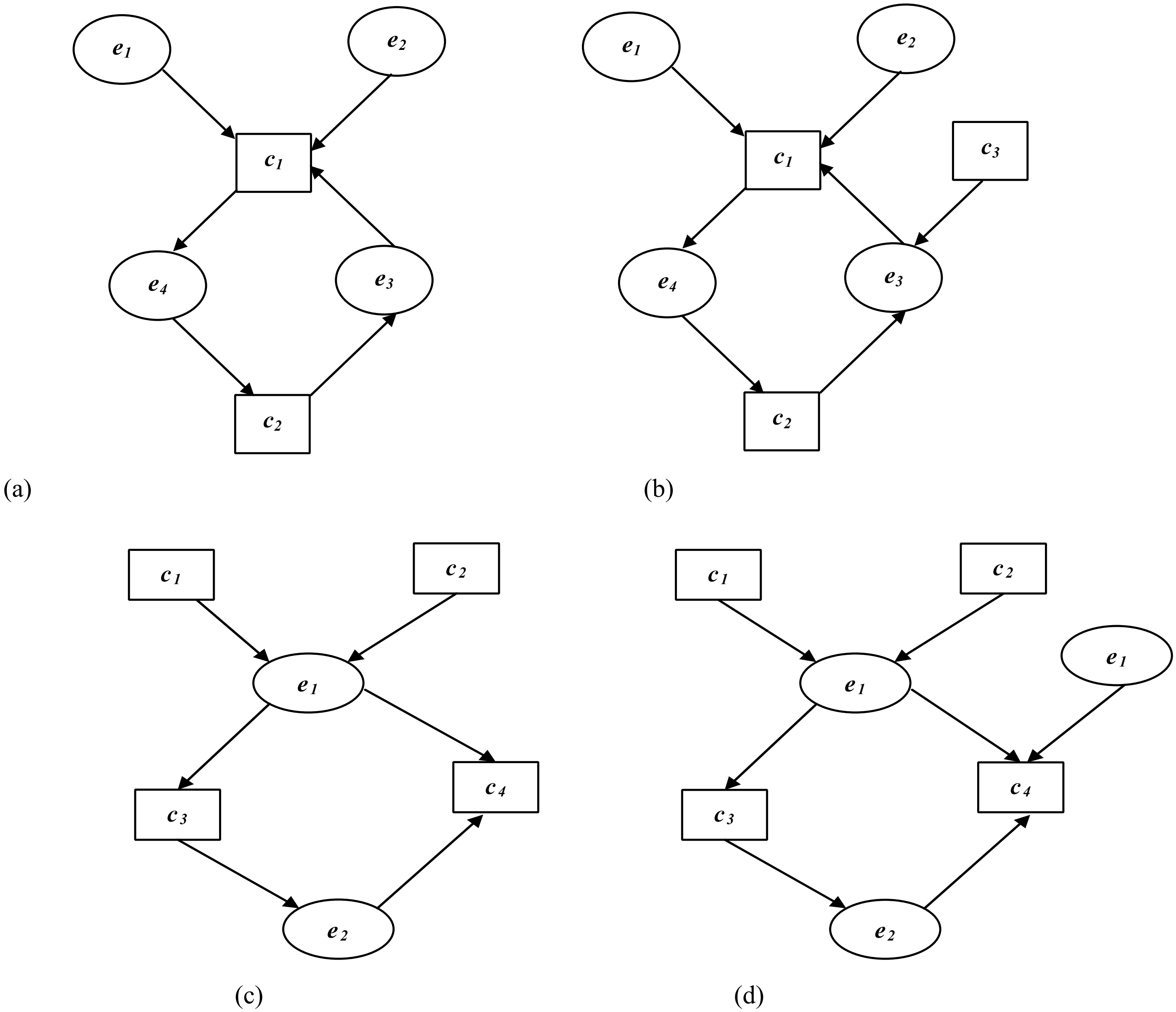

Figure 1: Different cases of Attack Graph Cycles considered.

In Figure 1(a) the cycle can be reached through a conditional node c_1. This node can be achieved by any of e_1, e_2, e_3 exploit nodes. Exploit node e_4 depends on c_1, c_2 depends on e_4 and e_3 depends on the conditional node c_2. This implies that, if condition node c_1 is satisfied, e_4 and c_2 can be reached as they depend on c_1. If c_2 can be reached then e_3 can be exploited successfully. This example of Attack Graph cycle cannot be removed as all the attack nodes and privileges can be reached during attack. Figure 1(b) shows similar cycle of (a) only that the conditional node c_3 now depends on attack node e_3. The cycle can be reached through a conditional node c_1. c_1 is an OR-node which can be achieved by any of the exploit nodes e_1, e_2, e_3 .

Attack node e_3 requires both node c_2 and c_3 before it can be exploited successfully. The two attack nodes in the cycle can also be reached as they both depend on c_1. This type of cycle cannot be removed from the Attack Graph. Figure 1(c) presents a different case of Attack Graph cycle. The cycle can be reached through the attack node e_1. e_1 is an AND-node which requires predecessors c_1, c_2 and c_4 to be satisfied before it can be successfully exploited. c_3 and e_2 depend on attack node e_1 and c_4 also depends on e_2 . It can be seen that both nodes e_1 and c_4

depend on each other, and therefore, they cannot be reached during attack. This type of cycle can be removed in an Attack Graph. 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 requires all nodes c_1, c_2 and c_4 to be satisfies before the attack can be exploited. The conditional node c_4 requires either e_2 or e_3 as precondition before it can be achieved. Nodes e_2 and c_3 all depend on the exploit node e_1, however, c_4 is an OR-Node which can either be influenced by external exploit node e_1 without depending on e_2. This means the dependency that exists between nodes c_4 and e_1 can be broken by exploiting the external node e_3. Thus, all the privilege and attack nodes can be reached during an attack. This cycle cannot be removed from the Attack Graph.

Thus, Attack Graphs naturally contain cycles. Modeling Attack Graphs models requires the removal of all available cycles present in an Attack Graph. In existing systems, not all the cycles in an Attack Graph can be removed completely. Those cycles that were discovered but could not be removed due to the reachability of attack nodes or privileges still constitute a directed cycle in the generated Attack Graph. The presence of any non-removable cycle in an Attack Graphs makes the Attack Graph, a directed cyclic graph. This type of graph is not suitable for the Attack Graphs modeling. Thus, in this paper, an attempt was made to formulate a novel method of identifying and handling all the available cycles in any given Attack Graph and converting then to a Directed Acyclic Graph (DAG).

4.Proposed Approach

The proposed approach in this paper leverages on graph theories. The approach can be divided into 3 parts: Generation of Attack Graph, Identification and removal of cycles in the Attack Graph and the Determination of valid attack paths from the Attack Graph. The detailed flowchart diagram of the proposed approach i.e process of generating the Attack Graph and scaling the generated Attack Graph is presented in Figure 2, the details are further described in the procedure presented as follows.

i. OAUNET was selected as a problem domain (i.e.Network environment).

ii. The network connection mapping and domain knowledge of the vulnerability information were identified using Nessus scanner tool. The Network connection mapping entailed information available across all the hosts in the target network. This includes information about the topology or connectivity (unique host identifiers like the host IP and host name), services running on the hosts and the available vulnerabilities with respect to the operating systems, software and services that have security flaws in the network hosts. Also, the domain knowledge of the vulnerability information was identified using NVD. This presents the dependency or relationship among the different vulnerabilities that exist in the target network. These are usually the pre and post conditions of each vulnerability of the hosts identified within the network

iii. The Nessus vulnerability scanning report was exported as .nessus file. The MulVAL takes in the Nessus scanning result (.nessus) as input, which is then translated into MulVAL datalogs.

iv. The Attack Graph was generated using MulVAL framework, the details were treated offline in this paper.

v. All cycles in the generated Attack Graph were detected and decision was made whether to remove them or not using the proposed Cycle handling Algorithm in Figure 3 (Cycle Detection and Handling). vi. The Attack Graph was scaled using the proposed Valid Path Algorithm in Figure 4 (Attack Graph Scaling).

4.1 Proposed Cycle Handling Algorithm

An Attack graph is a directed cyclic graph (DCG). It contains some set of strongly connected components where subsets of the vertices (exploits and conditions) are strongly connected to one another. Johnson (1975) presented an algorithm that can detect all the possible cycles in a directed graph. The detection of Attack Graphs‟ cycles will be an improvement of this algorithm.The improved algorithm presented in Figure 3 shows how cycles in Attack Graphs can be handled. This algorithm takes the directed cyclic Attack Graph as input and treats the cycles in the Attack Graph as strongly connected components. Each cycle in the set of strongly connected components found in the attach graph is subjected to exploit reachability. This is important to determine if the cycle is removable or not.

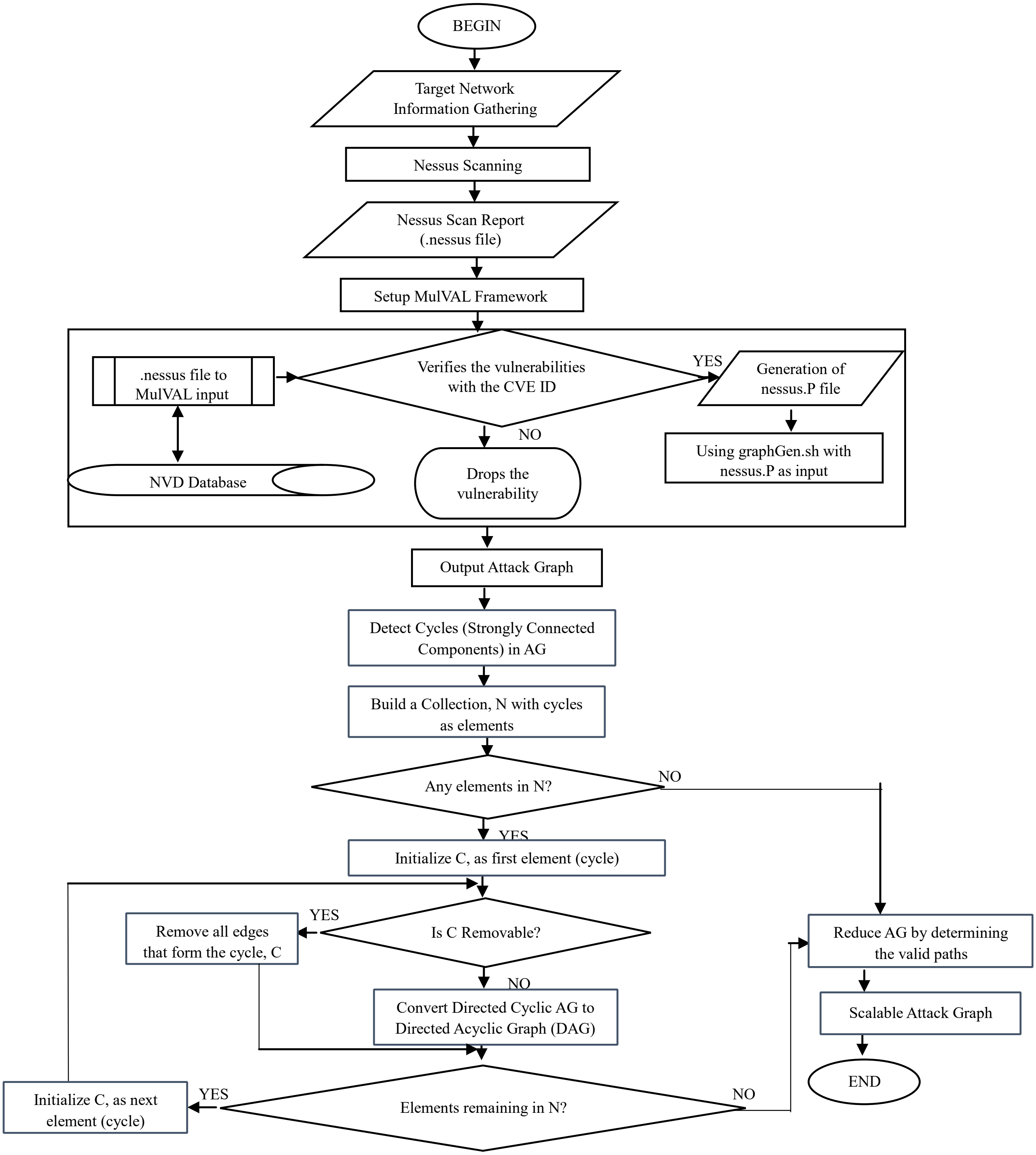
BEGIN
Target Network Information Gathering
Nessus Scanning
Nessus Scan Report (.nessus file)
Setup MulVAL Framework
.nessus file to MulVAL input
Verifies the vulnerabilities with the CVE ID
YES
Generation of nessus.P file
NO
NVD Database
Drops the vulnerability
Using graphGen.sh with nessus.P as input
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: The Flowchart of the proposed method

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

4.2 Proposed Valid Attack Path Detection Algorithm

The Attack Graphs generated by previous approaches do not scale as the size of the nodes in a network system increase. A node could have hundred number of vulnerabilities, which couldalso constitute into the exponential growth of the Attack Graph. In an enterprise network, such Attack Graphs are difficult to visually interpret for the purpose of security risk management of the network system. In order to address these scalability issues of generated Attack Graphs, this research proposed a forward search based algorithm in Figure 4 that identifies the valid attack paths from the Attack Graphs. The valid attack paths are easier and faster to comprehend that the generated Attack Graphs.

```
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.

5 Result And Discussion

The results of the proposed scalable Attack Graph generation model is presented as follows:

5.1 Attack Graph Generation

Figure 5 shows the generated Attack Graph with 103 nodes. it was rendered with numeric values assigned to each node for better visualization. The AND nodes of this MulVAL Attack Graph are shaped as ellipses while the OR nodes are in diamond shape and the vulnerability nodes are in boxes. The leaf nodes are the configurations on each host of the network system which usually have no ancestor. Upon critical observation of the Figure 5, it was noted that the generated Attack Graph is quite big and really too complex to understand.

5.2 Directed Acyclic Graph Generation

The implementation of the algorithm in Figure 3 was done with JAVA programming language using Netbeans IDE 8.0.2 running on Java Development Kit (JDK) 1.8.0 Update 25.

The algorithm was tested with the generated Attack Graph in Figure 5. There are 153 cycles in this Attack Graph. All the cycles found in this Attack Graph are listed in the Appendix Unfortunately, all of these cycles are not removable.. The Attack Graph contained 7 edges in the feedback arc set. This set of edges comprises of edges that could be removed in the Attack Graph in order to produce a directed acyclic graph. The list of the 7 edges is presented in Table 1. The removal of these edges produced the required Directed Acyclic Graph presented in Figure 6.

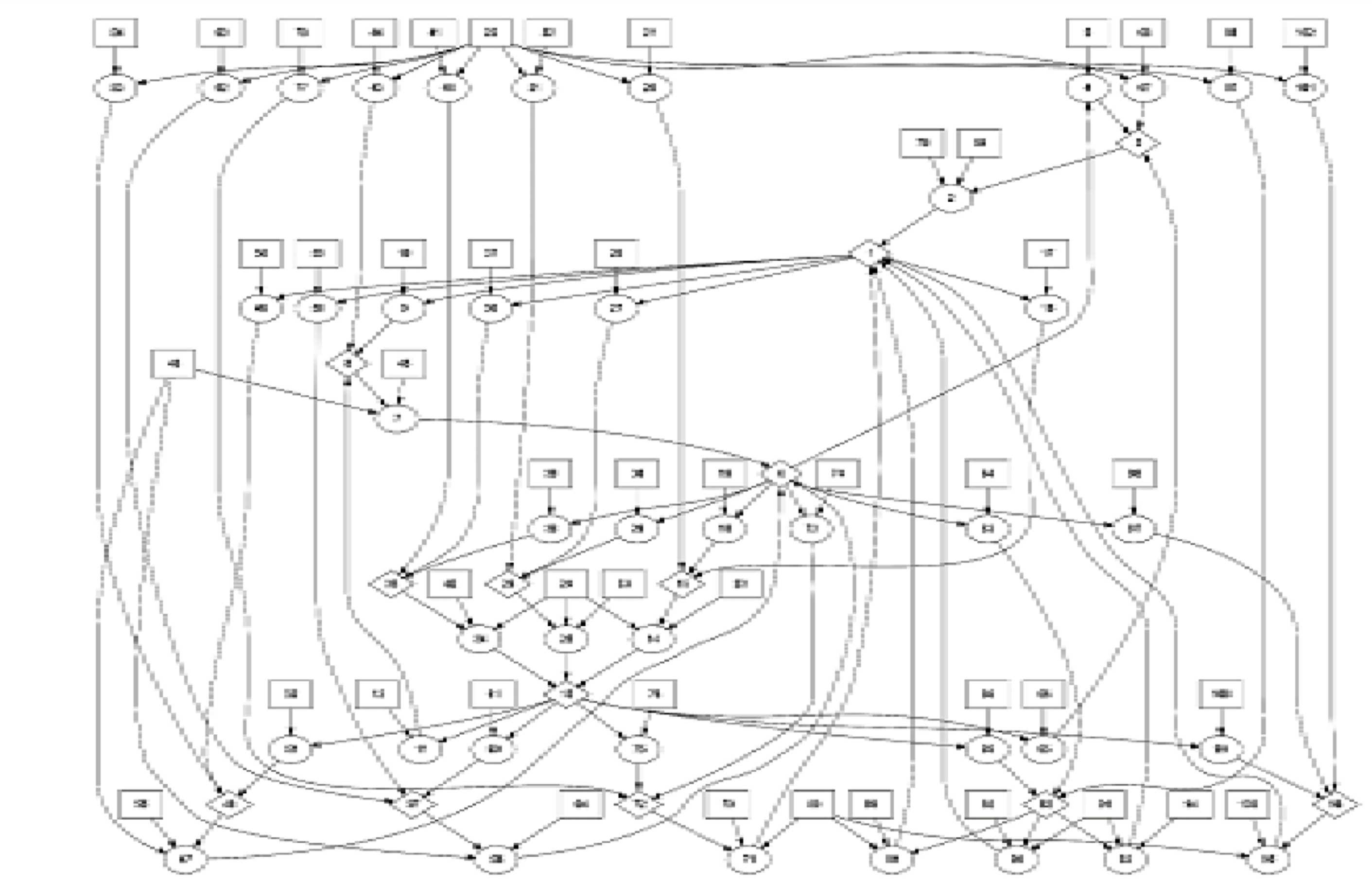

Figure 5: The Generated MulVAL Attack Graph (Node numbering) Table 1: The Feedback arc Set of the generated Attack Graph

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 algorithm in Figure 4 takes the Directed Acyclic Graph, the initial attacker‟s condition and the final attacker‟s goal as input. The output of this algorithm is a set of valid paths which formed the Attack Graph needed for the security risk management. The implementation of this algorithm was tested using the Directed Acyclic Attack Graph presented Figure 6.

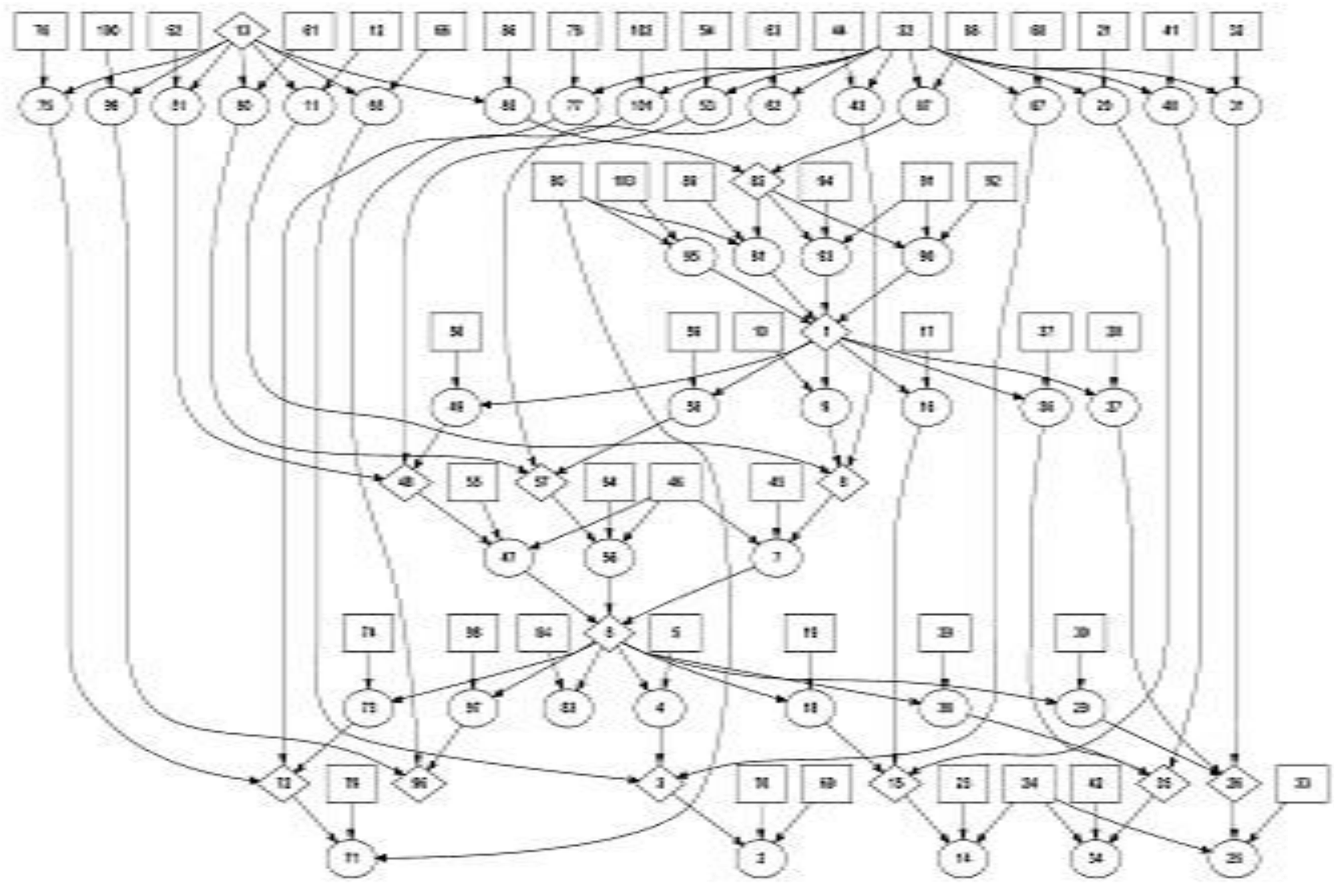

Figure 6: Generated Directed Acyclic Attack Graph

The initial attacker condition is node 22, which presents the attacker‟s locating the Internet for remote network exploit. Nodes 1, 6 and 13 are the final attacker‟s goal (to execute some exploit codes on each target host). This work assumes an attacker exploits each of the final goals independently. Figure 7 presents the result of the implementation of this algorithm with Figure 7(a) and Figure 7(b) showing the valid paths using node 1 and 6 as the final attack goalsrespectively. However, there is no valid path from the attacker‟s initial condition goal (node 22) and attack goal on node 13.

Figure 8 presents the final Attack Graph which was generated by merging the sub-graphs in Figure 7(a) and (b). In terms of the overall size of the graph, it can be seen that the final output is more scalable and easier to interpret or comprehend than the one generated in Figure 5. In addition, the MulVAL Attack Graph in Figure 5 has a total logical size of 2.29MB, the directed acyclic Attack Graph presented in Figure 6 is lighter with a total size of 1.86MB. The enhanced Attack Graph generated from this study as presented in Figure 8 has a total logical size of 0.97MB.

6. Summary and Conclusion

This paper has presented a graph-theoretic approach to addressing the scalability issues of Attack Graphs. An algorithm was formulated to detect and handle the cycles that are always present in Attack Graphs. These cycles can be removed or not, depending on if the exploit node in the cycle can be reached by an attacker. This paper also present a valid attack path detection algorithm which can be used to determine the most critical and valid paths of an Attack Graph.The proposed approach will enhance security assessment of network systems that are visually dependent on Attack Graphs.

This reduces the scalability problem of such Attack Graph which grows exponentially

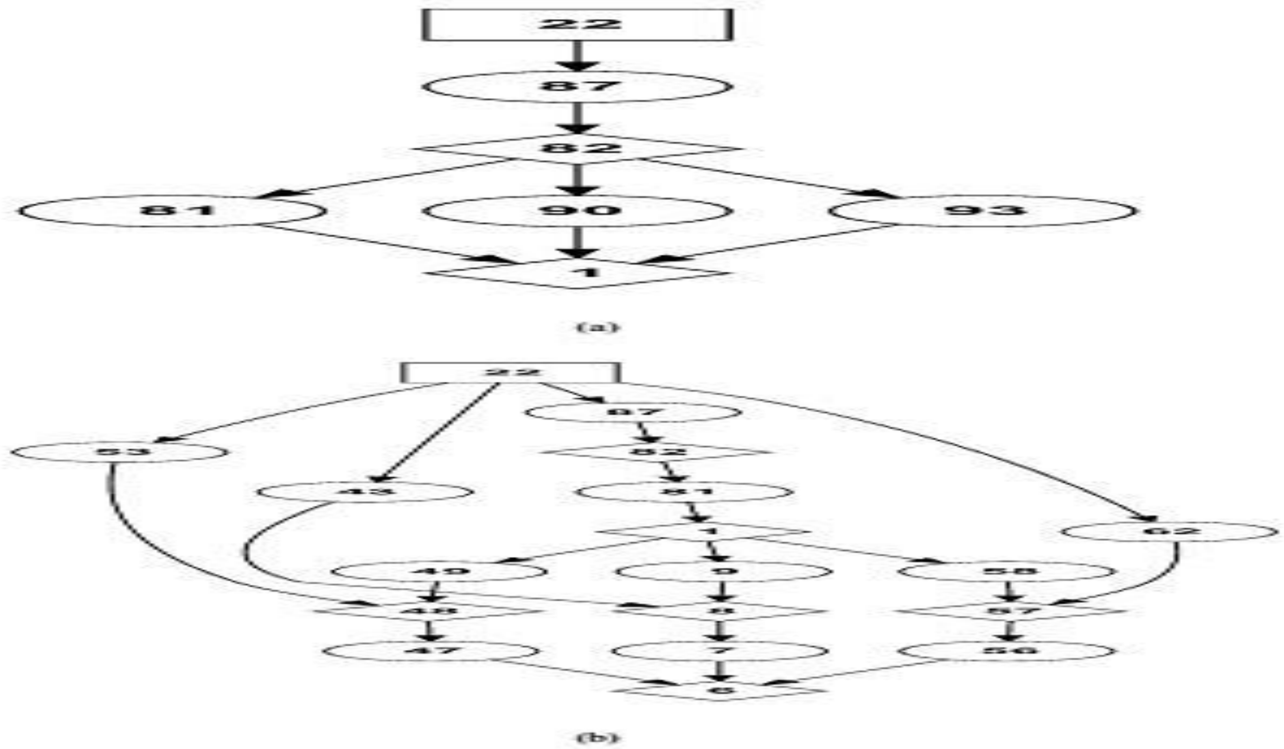

Figure 7: Valid Attack Paths from initial attacker's condition

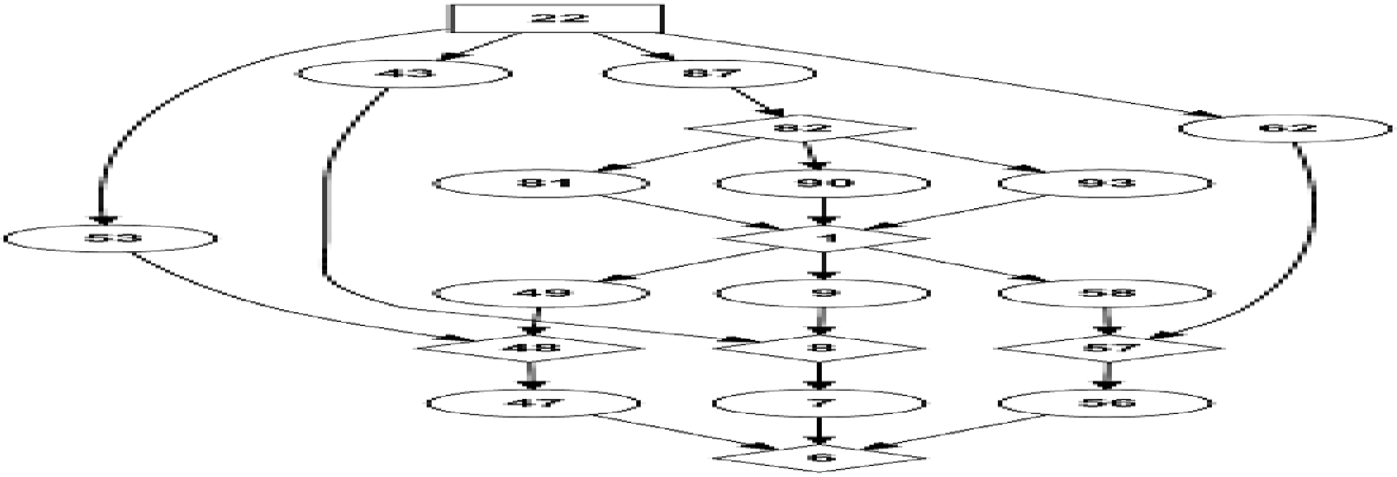

Figure 8: Enhanced Attack Graph

when the size of the network hosts and vulnerabilities increase. Thus, it will enable fast decision making by the network managers during such vulnerability assessment.

8. References

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.

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

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.

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

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.

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

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.

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

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

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.*

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.

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

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

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.

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,

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

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.*

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

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.

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.

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.

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

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*

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.