www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 5 Issue 5 May 2016, Page No. 16532-16538

Attack Graph-Based Approach For Enterprise Networks Security Analysis

Saidu Isah Rambo, Ibrahim M, Anka

ABSTRACT

Network administrators are always faced with numerous challenges of identifying threats and in retrospect, securing the organization's network. The classical approach of identifying the vulnerability in the network is by using commercially developed tools that do not take into cognisance vulnerability interaction between network elements and their behavioral pattern.Therefore, network administrators have to take a hollistic methods to identify vulnerability interrelationships to be captured by an attack graph which will help in identifying all possible ways an attacker would have access to critical resources. The objective therefore is to design an attack graph–based approach for analyzing security vulnerabilities in enterprise networks, implement and evaluate performance of the approach.

This work proposes an attack graph network security analyser based. The attack graph directly illustrates logical dependencies among attack goals and configuration information. In the attack graph, a node in the graph is a logical statement and an edge in the graph is represented by causality relation between network configurations and an attacker's potential privileges. The benchmark is just a collection of Datalog tuples representing the configuration of the synthesized networks, the graph generation CPU time was compared to Sheyner attack graph toolkit. The result in the graph shows the comparison of the graph builder CPU time for the case of a fully connected network and 5 vulnerabilities per host which shows Sheyner's tools grows exponentially.Some important contributions of this work include establishing an attack graph–based approach for enterprise networks security analysis that can capture generic security interactions and specify security relevant configuration information.

KEYWORDS: Datalog, CERT/CC, FW, webServer, workStation, fileServer.

1.1 INTRODUCTION

The increased dependent and reliance of networks by enterprise cannot be over emphasized. The external as well as the internal, threats that are continually faced by these enterprises have always increased phenomenally. Network security administrators are always faced with numerous challenges of identifying these threats and in retrospect, securing the organization's network. The classical approach of identifying the vulnerability of each element in the network is by using commercially developed tools that do not take into cognisance vulnerability interaction between network elements. Additionally, it has to identify the behavioral pattern of individual elements in the network. Therefore, network administrators have to take a more proactive and hollistic approach to identify vulnerability interrelationships and possible interaction to be captured by an attack graph which would help in identifying all possible ways in which an attacker would have access to critical resources in the network.

Statistically, network administrators are most often faced with challenges as a result of software vulnerabilities on network hosts. For over 20 years, there have been an ever-growing number of security vulnerabilities discovered in software and information systems. According to the statistics published by CERT/CC in (2013); a central organization for reporting security incidents, the numbers of reported vulnerabilities have grown considerably in the last 10 years. It is expected that the rate at which new software vulnerabilities emerge will continue to increase, in the foreseeable future. With thousands of new vulnerabilities discovered each year, maintaining 100% patch level is untenable and sometimes undesirable for most organizations, while in many cases patches come right after vulnerability reports (William *et al.*, 2000).

2.1 MATERIALS AND METHOD

The systems require PC's with the following minimum configurations: The performance of the MulVAL scanner on a Linux 9 host (kernel version 2.4.20-8) was measured. The CPU B940 is an Intel(R) Pentium(R) processor with 4.0GB RAM. 40GB hard disk drive, A VGA monitor, A standard keyboard, a mouse and a converter/UPS to provide protection to the systems from excess power source.

The system is to be loaded with a unix/ linux software or the internetworks operating system of the Microsoft software and other relevant softwares.

There should also be an internet connection of atleast 3G internet link.

Routers and firewalls with different specifications are needed for the serial connections and can also be used for the LAN connections.

Switches, hubs, repeaters, bridges and vlans used in various connections are to be configured as appropriate to provide smooth communication.

Fibre optic cables, coaxial cables, STP cables UTP cables and various specifications of CAT 5 cables are needed to provide straightthrough cables, crossover cables and the rollover cables where applicable.

VSAT with a c-band is needed to direct communication with the satellite or the service provider.

3.1 SYSTEM DESING

A logical attack graph is a di-graph and can be represented in the form of tree with a possible cross links between nodes. Figure 3.1 shows both the graph representation of a logical attack graph. There are two kinds of nodes in the graph; the derivation node and the fact node. The derivation node is represented as a rectangle and the fact node is represented as a circle/small star. There are also two kinds of fact node; the primitive fact node represented as a solid small star and the derived fact node is represented as a circle with a number inserted or encribed inside.

Every fact node in a logical graph is label with a logical statement in a form of a predicate applied in its argument. The root node is the attack goal; in the example it is *exeCode(attacker,workstation,root)* meaning "the attacker can execute arbitrary code as user *root* on machine *workstation*" Every derivation node is label with an interaction rule that is used for the derivation step.

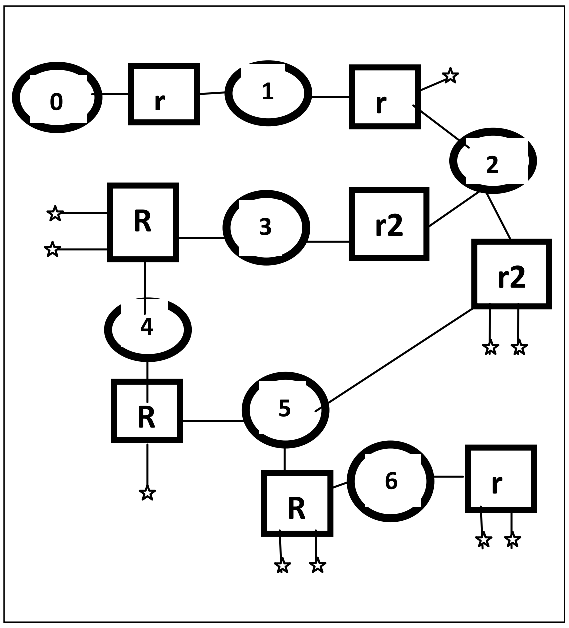

Fig 3.1: Graphical representation of logical attack graph

The edge in the graph represent the "depend on" relation. A fact node depends on one or more derivation node, each of which represent an application of an interaction rule that yield the fact; a derivation node depend on one or more fact nodes which together satisfies the precondition of a rule. Thus, a logical attack graph is a bipartite di-graph. The derivation nodes serves a medium between a facts and its 'reasons', i.e. how the fact becomes true. The derivation nodes directed from the fact node forms a disjunction. A derivation node represent a successful application of an interaction rule where all its preconditions are satisfied by its children. Therefore, the fact nodes directed from a derivation node forms a conjunction.

3.2 Algorithm

The proposed attack graph directly illustrates logical dependencies among attack goals and configuration information. The reasoning engine was modified so that besides a "true' or "false" answer, a Prolog query also records an attack simulation trace as a side effect of the evaluation and translated into the following form:

```
ExecCode (Attacker, Host, User)          :-
networkservice (Host, Program,
Protocol, Port, User)
vulExists (Host,Vul ID, Program,
remoteExploit, PrivEscalation),
netAccess (Attacker, Host, Protocol, Port),
assert_trace(because (
 'remote exploit of a server program',
 execCode(Attacker, Host, User) ,
[networkservice (Host, Program,
 Protocol, Port, User),
 vulExists (Host, Vul ID, Program,
 remoteExploit, PrivEscalation),
 netAccess (Attacker, Host,
 Protocol, Port)])).
```

A sub-goal is added, which calls the function **assert_trace.** When the evaluation of the rule succeeds, this function records the successful derivation into a trace file. In the attack graph, a node in the graph is a logical statement and an edge in the graph is represented by causality relation between network configurations and an attacker's potential privileges. A logical attack graph was viewed as a derivation graph for a successful Prolog query. In Prolog query, there is a derivation node "and" node, where all it children are conjuncted. A derived fact node is an "or" node where all it children represent different ways to derive and the primitive fact node is a leaf node in the graph which represent a pieces of configuration information.

Let (N_r, N_p, N_d , E, L, G)represents a logical attack graph, where N_r, N_p, *and* N_d and G are sets of derivation nodes, primitive fact nodes, derived fact nodes and attacker's goal respectively and are also referred to as disjoint nodes in the graph;$E \subset (N_r \times (N_p \cup N_{d,})) \cup (N_d \times N_r)$, L is the mapping from a node to its label and $G \in N_d$ is the attacker's goal. Also, Let T, I, C, F represent the trace step terms (interaction rule, fact and conjunct) and attacker's goal G, interactionRule (a string associated with interaction rule), conjunct (an instantiation fact or list of fact of interaction rule) and predicates (list of constant) respectively. A fact is primitive if it comes from the input to the MulVal-reasoning engine. A derived fact is the result of applying interaction rule iteratively on the input facts.
The attack graph is obtained as follows:

DOI: 10.18535/ijecs/v5i5.33

```
Let $N_r$, $N_p$, $N_d$ , E, L $\leftarrow$ 0;
For each $t \in T$, the derivation node $r$ is
$N_r \leftarrow N_r$ U {r};
L $\leftarrow$ L U {r $\rightarrow$ I};
For n $\epsilon$ $N_d$ suchthat L{n} = F then
E $\leftarrow$ E U {(n, r)
Otherwise, a new fact node n is created as
L $\leftarrow$ L U {r $\rightarrow$ Fact};
$N_d \leftarrow N_d$ U {n};
For each fact f in C
For c $\epsilon$ ($N_p$ U $N_{d)}$ such thatL(c) = f then
E $\leftarrow$ E U {(r, c)}
Otherwise, a new fact node c is created as
$L \leftarrow L$U {$c \rightarrow f$}
```

A logic attack graph can be constructed from the trace step information. Every trace step term becomes a derivation node in the attack graph. The fact field in the trace step becomes the node's parent and the conjunct field becomes its children. The performance evaluation of the approach was based on the computational complexity of the new approach with the existing one. The generation of attack trace only introduces a constant time overhead for every successful Prolog engine derivation.

3.3 Logical/automated attack-graph generation

While attack trees generated by the meta-interpreter serve the purpose of visualizing attack paths, the methodology also has several setbacks. Meta-interpreting; a prolog program is one order of magnitude slower than executing it directly in Prolog. Moreover, even if there is only a polynomial number of facts that can be derived by a Datalog program, the number of proof/attact trees generated could be exponential in the worst case. The XSB system includes a justifier program (Bernstein et al., 2000) that can compute evidence of derived literals while the program is running, thus eliminating the need for meta-interpreting and repeatation which might eventually result in looping. The evidence is stored in the Prolog database and can be extracted for visualization. According to test results (Bernstein et al., 2000), online justification only introduces 8% runtime overhead to the program, much better than meta-interpretation. To avoid the exponential blow up of proof trees, an acyclic graph can be out-putted visualizing the logical relationships among derived literals. The size of such graphs is polynomial to the size of the program. For the attack graph generated to be more useful and user-friendly attack graph generating software was developed.

4.1 LOGICAL TRACE STEP

Figure 4.4 shows the number of attack simulation trace steps, which is the inputed to the graph builder, is shown for the same set of test cases. For the worst case scenario, the number of trace steps is a quadratic function of the number of hosts. This verifies that Datalog evaluation inthe reasoning engine takes O(n2) derivation steps to complete.

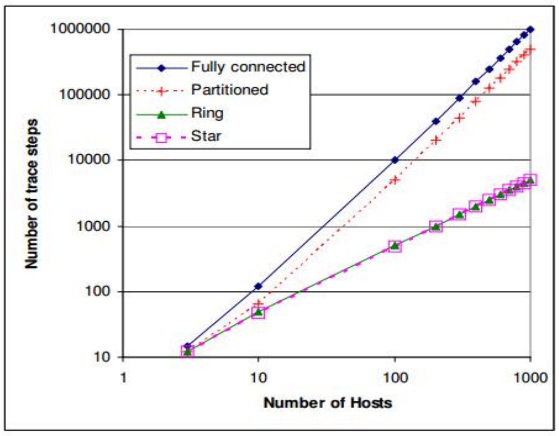

Figure 4.1: Graph representing trace steps

4.2 CPU Usage

Figure 4.1 shows the graph generation CPU time for each of the simulated analysis

DOI: 10.18535/ijecs/v5i5.33

problems of various sizes and topologies. The worst case is for a fully connected network. In this case the asymptotic CPU time is between O(n2) and O(n3), where n is the number of hosts. In Figure 4.2, it is noted that ideally the complexity is O(n2), if table look-up takes constant time. However, our implementation uses the simple "map" template in Java standard library and its look-up time depends on the size of the table. It is opined that after replacement with a custom-designed hash table implementation, the graph generation time would be near quadratic even for the worst case scenario.

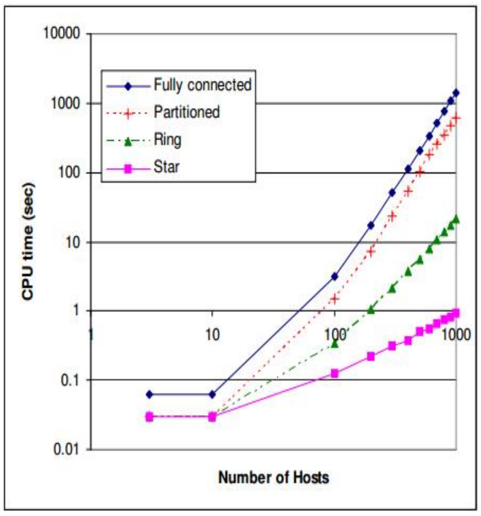

Figure 4.2: Graph generation CPU usage as a function of network size for several network topologies.

4.3 RAM usage

Figure 4.3 showsthe memory usage as a function of network size for the same four network topologies. The worst case here is again a fully connected network, which has a asymptotic memory usage slightly lower than O(n^2). In the two biggest cases (1000 host for fully-connected and partitioned network), we almost exhausted the 1GB memory on the test machine. The memory usage for the "star" and "ring" topology are not identical, although the difference is not quite visible on logarithmic scale.

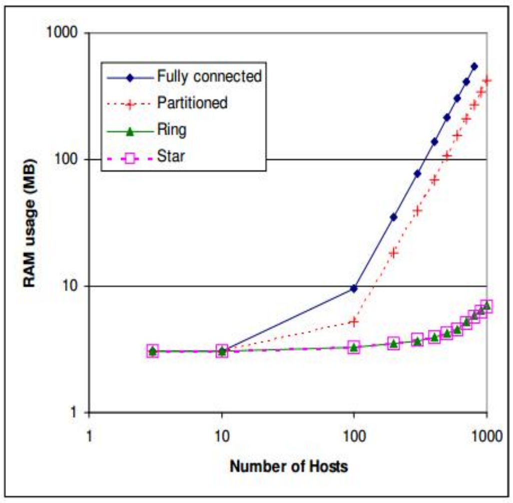

Figure 4.3: **Graph generation memory usage as a function of network size for several network topologies.**

Figure 4.4shows that the number of derived fact nodes in the attack graph grows linearly with the size of the network. An interesting case is the one for the "star" topology where the graph nodes remain constant regardless of the network size. This is because in that topology, the only attack path is from the attack machine to the hub, and then from the hub to the target machine.

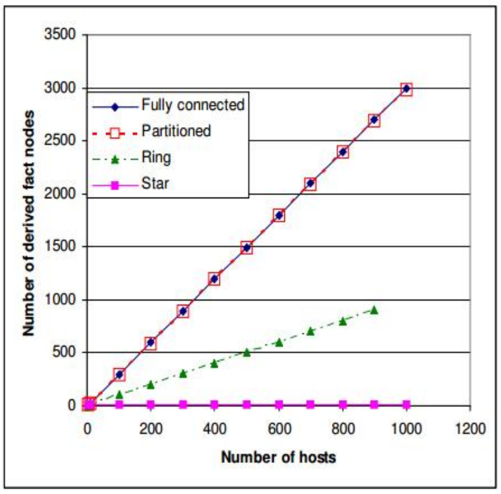

Figure 4.4: Graph representing derived facts

Figure 4.5 Showsthe attack graph generation CPU time is shown as a function of the network size for a fully connected network and for the number of vulnerabilities per host varied from 1 to 100. It shows that vulnerability density has a bigger impact when the network size is small. As the network size grows the CPU time is dominated by the number of machines, and thus vulnerability density has a less visible impact. Our graph builder was directly compared to the Sheyner attack graph toolkit by running both tools with equivalent input data. The Sheyner attack graph toolkit was tested on a Pentium III-M CPU, 256MB RAM, Fedora Core 1 LINUX operating system.

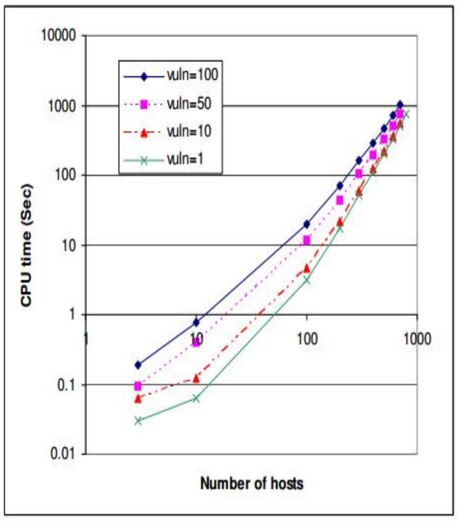

Figure 4.5: Graph generation CPU time

Figure 4.5: Graph generation CPU time for a fully connected network and number of vulnerabilities per host varying from 1 to 100.

Figure 4.6 is a comparison of graph builder CPU time for the case of a fully connected network and 5 vulnerabilities per host (note that only the Y axis is on logarithmic scale in this chart). From the diagram it is clear that the running time for Sheyner's tool grows exponentially. The growth trend for MulVAL is not obvious in this diagram because the running time is too short. But the difference between the two tools is obvious.

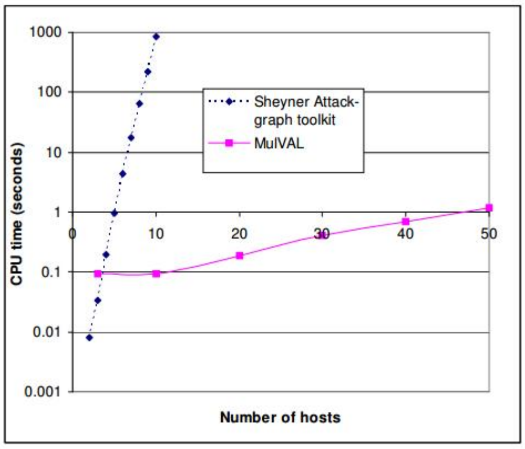

Figure 4.6 Graph generation CPU time

Figure 4.6Graph generation CPU time compared to Sheyner attack graph toolkit. Fully connected network and 5 vulnerabilities per host.

5.0 CONCLUSION AND RECOMMENDATIONS

5.1 CONCLUSION

The ultimate objective of this paper is to The specific objectives are to: design an attack graph–based approach for analyzing security vulnerabilities in enterprise networks; andimplement and evaluate performance of the approach.

5.2 RECOMMENDATIONS: It is recommended that It would be necessary for further study to be carried out in the areas of Also, the patching of machines in a network systems does not require rebooting the system. It would be an interesting research topic to study how to provide automatic and a heuristics as to what countermeasures to apply. This is beyond the scope of this dissertation and is left for further research work.

Finally, there is need to investigate precisely how to handle aggregates and negation in a distributed setting. It seems to be possible to incorporate well-known techniques to maintain states in the centralized setting into PSN[v]

References

Roychoudhury A., Ramakrishnan C. R., and Ramakrishnan I. V. (2000). Justifying proofs using memo tables. In Principles and Practice of Declarative Program- ming, pages 178–189.

Konstantinou A.V., Yemini Y. and Florissi D. (2002). Towards self-configuring networks. In DARPA Active Networks Conference and Exposition (DANCE), San Franscisco, CA.

Gelder A., Ross K. and Schlipf J.S. (1988). Unfounded sets and well- founded semantics for general logic programs. In PODS '88: Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 221–230, , ACM Press. New York, NY, USA.

Keromytis A.D., Ioannidis S., Greenwald M.B., Smith J.M. (2003). The STRONGMAN architecture. In Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX III), pages 178 –188, Washington, DC.

Wang A., Jia L., Liu C., Loo B.T., Sokolsky O. and Basu P. (2009). Formally Verifiable Networking. In SIGCOMM HotNets-VIII.

Gupta A., Mumick I.S. and Subrahmanian V. S. (1993). Maintaining views incrementally. In Peter Buneman and Sushil Jajodia, editors, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, pages 157–166. ACM Press.

Loo B.T., Condie T., Hellerstein J.M., Maniatis P., Roscoe T. and Stoica I. (2005). Implementing Declarative Overlays. In SOSP.

Loo B.T., Hellerstein J.M., Stoica I. and Ramakrishnan R. (2005). Declarative Routing: Extensible Routing with Declarative Queries. In SIGCOMM.

Loo B.T., Condie T., Garofalakis M., Gay D.E., Hellerstein J.M. Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. (2006). Declarative Networking: Language, Execution and Optimization. In SIGMOD.

Loo B.T., Condie T., Garofalakis M., Gay D.E., Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T. Stoica I. (2009). Declarative Networking. In Communications of the ACM (CACM).

Loo B.T., Zhou W. (2012) NDlog Declarative Networking declarative query language for specifying and implementing network protocols, Morgan & Claypool Publishers, Washington.

Schneier B., Wiley J. (2000) Secrets & Lies: Digital Security in a Networked World, chapter 21.