© 2014 IJIRT | Volume 1 Issue 4 | ISSN : 2349-6002

NETWORK SECURITY USING GRAPH THEORY

Sweety Sen, Sonali Samanta
B.Tech, Information Technology,
Dronacharya College of Engineering, Gurgaon , India

Abstract- **Network monitoring is a primary requirement for any network security. For monitoring network activities, we present the concept of Traffic Dispersion Graphs which can help easy identification of access patterns over a network. We also define an adjacency matrix attack graph to analyze and locate potential risks to protect critical network systems against multi step attacks. We suggest optimal solutions and configurations to next generation malware filter, based on graph-theoretic concepts to assess the importance of individual routers within the network, given a traffic pattern.**

I. INTRODUCTION

In today's globalized world, each and every activity is interlinked in one way or the other. Through the course of this paper we shall be analyzing computer networks, the flow of traffic or data from one computer to another and finally understand how this data can be at danger and how can it be saved? We assume every computer to be a node in a graph. The connections between two computers can be represented as an edge. The flow of data is in the direction of these edges. We can divide data into. A system of acknowledgment can be developed once a packet reaches a node i.e. a computer over the system.

II. HISTORY OF GRAPH THEORY

The origin of graph theory started with the problem of Koinsber bridge, in 1735. This problem lead to the concept of Eulerian Graph. Euler studied the problem of Koinsberg bridge and constructed a structure to solve the problem called Eulerian graph. In 1840, A.F Mobius gave the idea of complete graph and bipartite graph and Kuratowski proved that they are planar by means of recreational problems. The concept of tree, (a connected graph
without cycles[7]) was implemented by Gustav Kirchhoff in 1845, and he employed graph theoretical ideas in the
calculation of currents in electrical networks or circuits. In 1852, Thomas Gutherie found the famous four color
problem. Then in 1856, Thomas. P. Kirkman and William R.Hamilton studied cycles on polyhydra and invented the
concept called Hamiltonian graph by studying trips that visited certain sites exactly once. In 1913, H.Dudeney
mentioned a puzzle problem. Eventhough the four color problem was invented it was solved only after a century by Kenneth Appel and Wolfgang Haken. This time is considered as the birth of Graph Theory.

III. TRAFFIC DISPERSION GRAPHS

A major problem these days is keeping a check on the traffic and thus detecting applications that are not required. This is because many applications obfuscate their traffic using unregistered port numbers or payload encryption. In this paper, we propose the use of traffic Dispersion Graphs (TDGs) as a way to monitor, analyze, and visualize network traffic. TDGs model the social behavior of hosts ("who talks to whom"), where the edges can be defined to represent different interactions (e.g.

the exchange of a certain number or type of packets). With the introduction of TDGs, we are able to harness a wealth of tools and graph modeling techniques from a diverse set of disciplines. In this work, we propose a different way of looking at network traffic that focuses on network-wide interactions of hosts (as seen at a router. We argue that there is a wealth of information embedded in a TDG. For example, a popular website will have a large in-degree, while P2P hosts will be tightly connected. An edge can represent the exchange of at least one packet. In other words, a TDG can represent a particular type of interaction, which gives them significant descriptive power, as we discuss later in detail. TDGs can be seen as the natural next step in the progression of packet, flow, and host level aggregation. This is because a flow aggregates a set of packets, a host aggregates a set of flows originating and terminating at the host and a graph aggregates a group of hosts. Our main goal is to propose TDGs as a dioerent way of modeling traffic behavior, and show that they:

(a) have characteristic structure and provide visualizations that can distinguish the nature of some applications,

(b) describe traffic along new "dimension", the network-wide social behavior, which complements traffic characterization at the packet, flow and host levels.

TDG Formation: In this paper we focus on port-based TDGs. Throughout the paper and unless stated otherwise, when the legacy application for a port uses TCP, we use the EFSP edge filter on the corresponding destination port (e.g., TCP Port 25 for SMTP). When we examine UDP interactions, we use the EFP edge filter on the destination port of interest (e.g., UDP Port 53 for DNS). For ease of presentation, we will refer to each port-based TDG using the name of the dominant or well-known application under that port. For example, the HTTP TDGs is formed by using as edges all the TCP SYN packets that have as destination port the number 80. Since we use edge filtering by port number, the TDGs capture aspects of any application that uses these ports. We are fully aware that many nonstandard applications, such as P2P traβc, use standard ports such as Port 80. However, port-based filtering is consistent with our use of TDGs as a monitoring tool. For example, if at some point traffic at TCP Port 80 appears significantly different, it could be:

(a) a new benign or malicious application tunneling its traffic under that port, or

(b) a change in the behavior of the traditional application.

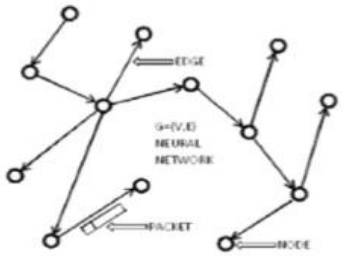

Figure 1 Traffic Dispersion Graph

TDG Visualization: Traditionally, visualization of traffic in monitoring tools has largely been limited to visualizing measures of traffic volumes on a per flow basis. By contrast, we show that TDGs lend themselves to simple graphical visualizations of interaction patterns. We can identify several distinctive structures and patterns in TDGs, which are indicative of the behavior of different applications. Node degrees - The degrees of various nodes and their connectivity in a TDG helps us in visually determining the type of relationship between the nodes.

Conclusions and Future Work

Two essential features in network monitoring tools dealing with vast amounts of network data are aggregation and the ability to spot patterns. TDGs represent a natural extension of previous approaches that have aggregated at the packet, flow, and host levels by aggregating across nodes. The aggregation across nodes also reveals patterns of social interaction across nodes that are specific to applications. These interaction patterns or graph structures can then be used to visually and quantitatively monitor existing applications and potentially detect concealed applications and mal-code. Assuming that not many diverse applications use the same port number, port-based TDGs can be used in order to identify the type of application utilizing a given port. We envisage such a system working as follows. First, given any type of edge filter (e.g., a port number) we first construct the TDG. Next, using graph metrics, we identify the nature of the application on that port (e.g., if is a

client-server, peer-to-peer, or malware application). The filter selection can be:
(a) extracted automatically, triggered by an anomalous behavior or
(b) given a priori by the network administrator, deviations can be used to trigger an alarm.

IV. INTRODUCTION TO ATTACK GRAPHS

Attack Graphs Generation: Several tools measure point-based vulnerabilities on individual hosts. However, vulnerabilities on a network being of causal relationships actually arouse more impact and damage to a whole network and persist longer and more undetectable if we are unable to defend against them in relevance. Attack Graphs of Automated Generation encode the causal relationships among vulnerabilities and tell whether critical assets are secure enough against potential multi-step combining attacks. The automated tool succeeds to automate the generation of attack graphs and releases administrators from error-prone and arduous manual work. Therefore, it has become a desirable tool for administrators to analyze their networks, report potential risks and protect their networked assets. But there is a limitation to it, that is, the complexity problem, regarding the size of the network and vulnerabilities that exist in the network. In practice, attack graphs always exceed human ability to visualize, and understand.

A Motivating Example: A network configuration shows connections between machines and vulnerabilities' distribution on a network. A type graph tells the dependency or exploits relations between vulnerabilities. Out of the two inputs, a vulnerability-based attack graph can be drawn out, in which a security-related vulnerability or condition represents the system state, and an exploit between vulnerabilities is modeled as a transition. Figure illustrates a network configuration example. The left side is the network configuration graph. h1 is a machine

h1(v2,v3) h2(v1,v2)
h3(v1,v3)

Figure 2: Dependencies

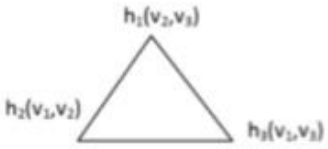

Figure 3: Machine

Having vulnerabilities v2 and v3 (these vulnerabilities are generalized with simplified notations, which do not express any concrete vulnerability but conceptual ones mainly for their relationships). h2 has vulnerabilities v1 and v2. h3 has vulnerabilities v1 and v3. The right side is a type graph that expresses dependent relations between vulnerabilities. v1 is the first vulnerability that is assumed satisfied on its own. v2 is dependent on the satisfaction of v1. v3 is dependent on the satisfaction of v2. Therefore, v1 is the pre-condition of v2; v2 is the precondition of v3. In another way, we can say v2 is the post-condition of v1 and v3 is the post-condition of v2. Here the satisfied or satisfaction means that vulnerability on a machine, whose preconditions have all been satisfied by an attacker, can be reached or acquired by the attacker now. Acquiring the vulnerability-based attack graph has many approaches. However, a direct way is to find all the attack paths, and then uses them to set up an attack graph.

$h_2v_1 \rightarrow h_1v_2 \rightarrow h_1v_3$	$h_3v_1 \rightarrow h_1v_2 \rightarrow h_1v_3$
$h_2v_1 \rightarrow h_1v_2 \rightarrow h_3v_3$	$h_3v_1 \rightarrow h_1v_2 \rightarrow h_3v_3$
$h_2v_1 \rightarrow h_2v_2 \rightarrow h_1v_3$	$h_3v_1 \rightarrow h_2v_2 \rightarrow h_1v_3$
$h_2v_1 \rightarrow h_2v_2 \rightarrow h_3v_3$	$h_3v_1 \rightarrow h_2v_2 \rightarrow h_3v_3$

Figure 4: Attack Paths

Adjacency Matrix Clustering: The rows and columns of an adjacency matrix could be placed in any order, without affecting the structure of the attack graph the matrix represents. But orderings

that capture regularities in graph structure are clearly desirable. In particular, we seek orderings that tend to cluster graph vertices (adjacency matrix rows and columns) by common edges (non-zero matrix elements). This allows us to treat such clusters of common edges as a single unit as we analyze the attack graph (adjacency matrix). In some cases, there might be network attributes that allow us to order adjacency matrix rows and columns into clusters of common attack graph edges. For example, we might sort machine vertices according to IP address, so that machines in the same subnet appear in consecutive rows and columns of the adjacency matrix.Unrestricted connectivity within each subnet might then cause fully connected (all ones) blocks of elements on the main diagonal. In general, we cannot rely on a priori ordering of rows and columns to place the adjacency matrix into meaningful clusters. We therefore apply a particular matrix clustering algorithm [23] that is designed to form homogeneous rectangular blocks of matrix elements (row and column intersections). Here, homogeneity means that within a block, there is a similar pattern of attack graph edges (adjacency matrix elements). This clustering algorithm requires no user intervention, has no parameters that need tuning, and scales linearly with problem size. This algorithm finds the number of row and column clusters, along with the assignment of rows and columns to those clusters, such that the clusters form regions of high and low densities. Numbers of clusters and cluster assignments provide an information-theoretic measure of cluster optimality. The matrix clustering algorithm is based on ideas from data compression, including the Minimum Description Length principle [40], in which regularity in the data can be used to compress it (describe it in fewer symbols). Intuitively, one can say that the more we compress the data, the better we understand it, in the sense that we have better captured its regularities.

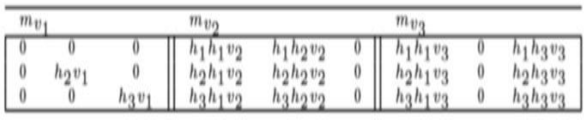

Figure 5: Adjacency Matrix Clustering

Matrix Operations for Multi-Step Attacks: The adjacency matrix shows the presence of each edge in a network attack graph. Taken directly, the adjacency matrix shows every possible single-step attack. In other words, the adjacency matrix shows attacker reachability within one attack step. As we describe later, we can navigate the adjacency matrix by iteratively matching rows and columns to follow multiple attack steps. We can also raise the adjacency matrix to higher powers, which shows multi-step attacker reachability at a glance. For a square (n × n) adjacency matrix A and a positive integer p, then A^p is A raised to the power p: In other words,

A^p = (A AA…. A) p times ….(1)

Here, matrix multiplication is in the usual sense. For example, an element of A^2

In Equation, the matching of rows and columns in matrix multiplication (index k) corresponds to matching steps of an attack graph. The summation over k counts the numbers of matching steps. Thus, each element of A^2gives the number of 2-step attacks between the corresponding pair (row and column) of attack graph vertices. Similarly, A^3 gives all 3-step attacks; A^4 gives all 4-step attacks, etc.

For raising a (square) matrix to an arbitrary power, we can improve upon naïve iterative multiplication. This involves a spectral decomposition [41] of A. An n × n matrix always has n Eigen values. These form an n × n diagonal matrix D and a corresponding matrix of nonzero columns V that satisfies the Eigen value equation AV = VD. If the n Eigen values are distinct, then V is invertible, so that we can decompose the original matrix A as

$$\mathbf{A = VDV^{-1}} \quad \ldots\ldots(2)$$

Here D is a diagonal matrix formed from the eigen values of A, and the columns of V are the corresponding eigenvectors of V.

It is then straightforward to prove that

$$A^p = VD^p$$

V^{-1}, via V-1V = I. This product VD^p

V^{-1} is easy to compute since D^p is just the diagonal matrix with entries equal to the p th power of those of D, i.e.,

$$D^p = \begin{bmatrix} d_1^p & 0 & \cdots & 0 \\ 0 & d_2^p & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & d_n^p \end{bmatrix}.$$

Figure 6: Diagonal Matrix from eigen values

***Attack Prediction*:** In our approach, we place detected intrusions within the context of predictive attack graphs based on known vulnerability paths. We first compute a vulnerability-based attack graph from knowledge of the network configuration, attacker exploits, etc. We then form the adjacency matrix A for the attack graph, perform clustering on A. We then compute either the transitive closure of A, or the multi-step reachability matrix. Then, when an intrusion alarm is generated, if we can associate it with an edge (e.g., exploit) in the attack graph, we can thus associate it with the corresponding element of any of the following:

- The adjacency matrix A (for single-step reachability).
- The multi-step reachability ma trix in Equation (6) (for multi-step reachability).
- The transitive closure of A (for all-step reachability). From this, we can immediately categorize alerts based on the numbers of associated attack steps. For example, if an alarm occurs within a zero-valued region of the transitive closure, we might conclude it is a false alarm, i.e., we know it is not possible according to the attack graph. Or, if an alarm occurs within a single-step region of the reachability matrix, we know that it is indeed one of the single-step attacks in the attack graph. Somewhere in between, if an alarm occurs in a p-step region, we know the attack graph predicts that it takes a minimum of p steps to achieve such an attack. By associating intrusion alarms with a reachability graph, we can also predict the origin and impact of attacks. That is, once we place intrusion alarm on one of the vulnerability-based reachability graphs, we can navigate the graph to do attack prediction.

The idea is to project to the main diagonal of the graph, in which row and column indices are equal. Vertical projection (along a column) leads to attack step(s) in the forward direction. That is, when one project along a column to the main diagonal, the resulting row gives the possible steps forward in the attack. We can predict attack origin and impact either (1) one step away, (2) multiple steps away with the number of steps distinguished, or (3) over all steps combined. Here are those 3 possibilities:

- When using the adjacency matrix A, non-zero elements along the projected row show all possible single steps forward. Projection also can be done iteratively, to follow step-by-step (one at a time) in the attack.
- When using the multi-step reachability matrix in Equation (6), the projected row shows the minimum number of subsequent steps needed to reach another vertex. We can also iteratively project, either choosing single-step elements only, or "skipping" steps by choosing multi-step elements.
- When using the transitive closure, the projected row sh ows whether a particular vertex can be subsequently reached in any number of steps. Here, iterative projection is not necessary, since transitive closure shows reachability over all steps. We see that projection along a column of a reachability matrix predicts the impact (forward steps) of an attack. Correspondingly, we can project along a row (as opposed to a column) of such a matrix to predict attack origin (backward steps). In this case, when one projects along a row to the main diagonal, the resulting column gives the possible steps backward in the attack. As before, we can predict attack origin using either the adjacency matrix, the multi-step reachability matrix, or the transitive closure matrix. Just as for forward projection, this gives either single-step reachability, multi-step reachability, or all-step reachability, but this time in a backward direction for predicting attack origin.

Conclusion

In this paper, we defined the adjacency matrix attack graphs, which are a novel concept in the visualization and generation of attack graphs and successfully avoid the complexity problem. In the light of its definition, we formalized the adjacency matrix attack graph-based probabilistic security metric with the extension definition concerning cycles. The advantage of our proposed approach is that it simplifies the visualization to human eyes, which replaces those cluttering edges of an attack graph. It separates the complexity of attack graphs into two fractions: the network connectivity property and the interactions among an exploit dependency attack graph. No matter how many machines in a network, the visualization or representation always is controlled within a certain number of vulnerabilities and exploits. The adjacency matrix attack graph also facilitates the

probabilistic computations without the exponential explosion, the complexity of which is within O(n ^{2}).

V. MALWARE IDENTIFICATION

Once a system has been identified with irregularities from the original work we can predict an malware is trying to attack the system. Computer networks have become an ubiquitous but vulnerable aspect of corporate, university, and government life. Yet the increased complexity of computer networks combined with the ingenuity of attacker's means that they remain susceptible to expensive attacks from worms, viruses, Trojans, and other malicious software, which we simply refer to as malware. Network traffic filtering is one of many security methods available tone work administrators. Network traffic filters provide protection by sampling packets or sessions and either comparing their contents to known malware signatures or looking for anomalies likely to be malware. Filtering capabilities have begun to be integrated into routers themselves, so as to reduce hardware deployment costs and to allow for more adaptive security. Future traffic filters are expected to be configurable, networked, and even autonomous. Our objective in this paper is to investigate the deployment and configuration issues of such devices within an optimization framework. Computer networks have become a ubiquitous but vulnerable aspect of corporate, university, and government life. Yet the increased complexity of computer networks combined with the ingenuity of attackers means that they remain susceptible to expensive attacks from worms, viruses, Trojan's, and other malicious software, which we simply refer to as malware [1]–[3]. Network traffic filtering is one of many security methods available to network administrators. Network traffic filters provide protection by sampling packets or sessions and either comparing their contents to known malware signatures or looking for anomalies likely to be malware. Filtering capabilities have begun to be integrated into routers themselves, so as to reduce hardware deployment costs and to allow for more adaptive security [4]. Future traffic filters are expected to be configurable, networked, and even autonomous. Our objective in this paper is to investigate the deployment and configuration issues of such devices within an optimization framework.

Model and Problem Formulations: There are a lot of goals and restrictions that a Computer network administrator's faces at the network security level and the financial or at technical cost of achieving that security level. We combine and express these constraints and objectives within the four malware filter placement problems evaluated in this paper. We consider a network of configurable, networked routers with traffic filtering capabilities which can be dynamically and remotely set by a centralized server. Some subsets of these routers are source routers and another (potentially overlapping and typically identical) subset is destination routers. Other routers are core routers. We do not explicitly consider the effectiveness of malware filters. We assume that filtered packets are marked so that we do not redundantly filter particular packets. In addition, we assume that the network administrator has full knowledge of the network traffic, possibly with some delay. Finally, we do not consider how the act of filtering malware will alter the quantity of traffic on a link or the quantity of malware at future routers because we assume that the proportion of malware in the network is relatively low. Although we will discuss here packet filtering, all of the developed theory and results also apply to the filtering of sessions.

Centrality Measures for Network Link Assessment: We introduce two new centrality measures within the context of communication networks. Traditional centrality measures, as described in [9], involve source-destination pairs, but each pair is weighted identically. A more relevant and accurate centrality measure would weight source-destination pairs according to the magnitude of traffic that travels between them. Moreover, traditional centrality measures consider every node to be a potential source and destination, but this is not the case for core routers. Therefore, we propose only considering those nodes that are in fact

sources and/or destinations (i.e. no core routers) in our centrality calculation. Traffic betweenness centrality (TBC) is betweenness centrality with the above two

changes. Let R be a set of all vertices in an undirected graph. Let S ⊆ R contain all the sources, D ⊆R be the set of all destinations, and P be the set of all source-destination pairs (s,d). The number of shortest paths between s ε R and d ε R is σ_{sd}. The number of these shortest paths that pass through

some r ε R is $\sigma_{sd}(r)$. Moreover, the amount of traffic between s and d is usd. TBC assumes that there are multiple shortest paths between a source and destination and that they are equally likely to be used. TBC of a router r, denoted by CTB(r), is then defined as the fraction of shortest paths of all source- destination pairs that pass through a particular router, with each source-destination pair being weighted by its traffic magnitude.

CTB(r):=X(s,d)ε P usd σ_{sd}(r) σ_{sd} …. (1)

If at least one shortest path for any source-destination pair passes through a given router r, then CTB(r) > 0.

A special case would be based on stress centrality and called traffic stress centrality (TSC). Here we do not assume

that all shortest routes are used with equal likelihood but rather that one is chosen. In this case if a node is on this selected shortest route, then σ_{sd}= 1, otherwise it is 0. TSC of a router r is then defined as :

CTS(r) :=X(s,d)εP u_{sd} σ_{sd} (r) ….(2)

There exists an intuitive but not immediately obvious relationship between the traffic centrality measures defined here and the actual traffic that passes through a router. Assume that the traffic between all source-destination pairs on this network is routed using either (a) a load-balancing shortest path routing scheme where all the packets sent from a source node to a destination one are equally likely to be delivered through multiple shortest paths

between them or (b) a simple shortest path scheme where all packets between the source and the destination nodes are delivered consistently through a single shortest route. Then, the amount of traffic on a router r is equal to the TBC measure in the case of routing scheme (a) and the TSC measure in the case of scheme (b). These relationships can easily be proved by comparing the definitions of traffic centrality measures with simple equations for traffic at routers under these routing schemes. Traffic centrality measures capture the importance of routers on a network, and hence are helpful when defining objective functions for malware filtering problems

VI. Conclusions And Future Work

We have studied malware filter placement problems from an optimization perspective. After drawing the connection between traffic-weighted centrality measures and traffic measurements at routers, we chose a convex cost objective involving centrality measures. The first optimization problem we considered involves minimizing this cost subject to sampling and effective sampling rate constraints, as well as a constraint on the amount of traffic that can be filtered network-wide. Next we studied the case where instead of placing a hard upper bound on the quantity of filtering, we assign a cost to filtering and minimize a sum of it and the cost metric derived earlier. We then minimized a different cost metric involving a sum of filter deployment and filtering costs less a utility measure under the same constraints. We found exact or approximate centralized and dynamic solutions to these optimization problems and simulated the resulting strategies. Network traffic data from the Abilene dataset was used in these simulations. We compared these strategies with benchmark approaches to network traffic filtering. The simulation results confirm that by applying optimization tools we can achieve lower costs in a variety of contexts and when traffic magnitudes change rapidly. There are several obvious extensions to

this work. The optimization problems developed here should be solved in a decentralized manner for increased reliability and security. Various update algorithms could be considered when evaluating decentralized solutions. A natural extension to this paper would involve incorporating filter effectiveness with Bayesian analysis. This would al- low for a comparison between signature-based and anomaly- based filters. Constraints on the amount of signature-based and anomaly-based filtering could be set. Finally, we are planning to use the Abilene data toper- form more thorough simulations with the realistic Network Security Simulator (NeSSi).

References

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based net- work vulnerability analysis. In Proceedings of the 9th ACM

Conference on Computer and Communications Security (CCS'02), 2002.

[2] R. Deraison. Nessus scanner, 1999. Available at http://www.nessus.org.

[3] D. Farmer and E. Spafford. The COPS security checker system. In USENIX Summer, pages 165–170, 1990.

[4] S. Jajodia and S. Noel. Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response.
[5] In B. Bhattacharya, S. Sur-Kolay, S. Nandy, and A. Bagchi, editors, Algorithms, Architectures, and Information Systems Security. World Scientific Press, 2007.
[6] P. Mell, K. Scarfone, and S. Romanosky. Common vulnerability scoring system. IEEE Security & Privacy Magazine, 4(6):85–89, 2006