Visualizing Graph Features for Scanning Attack Detection

Robert F. Erbacher
U.S. Army Research Laboratory
2800 Powder Mill Rd.
Adelphi, MD 20783
Robert.F.Erbacher.civ@mail.mil

Maggie Cheng   Quanmin Ye
Missouri University of Science and Technology
Dept. of Computer Science
Rolla, MO 65401
{chengm, qy4y4}@mst.edu

ABSTRACT
Detection of sophisticated network scans, such as low and slow scans, requires correlation of large amounts of network data over long periods of time. While challenging, such detections are critical as it is the sophisticated attacker employing these approaches that it is absolutely critical to detect. To detect such sophisticated scans we propose the integration of graph feature extraction techniques with visualization to simultaneously optimize computational complexity and human analyst time. Clearly, a fully automated approach will be desirable in the future, the proposed approach provides real world detection capabilities without excessive computation or analysis overhead. We propose a design of graph algorithms to extract the features/characteristics necessary for scan detection, employ visualization techniques to show the relevant multidimensional characteristics, and provide test cases that show that the proposed work enables simpler and faster detection than previous approaches.

1. INTRODUCTION
Intrusion detection has a long history, beginning with Anderson's paper [2]. To this day, the primary deployment of intrusion detection has focused on signature based techniques which match activity against specific rules for known malicious behavior. This is the case even though it was in 1986 that Denning [3] first proposed anomaly detection. Anomaly detection research has been pursued consistently since the time of Denning's paper [4]. However, the developed techniques have not proven effective due to low true positive rates and/or high false positive rates. The difficulty that has arisen is that advanced persistent threats are not amenable to signature based approaches. This has led to renewed interest in the development of anomaly detection approaches that will achieve deployable true positive and false positive rates, as per Axelsson [5]. Anomaly detection algorithms have been challenged due to the difficulty in defining what an anomaly is in the context of computer network activity. Given the lack of success of existing anomaly detection, an approach gaining interest is how to narrow the scope, provide anomaly detection with additional input, or have multiple techniques work together in an ensemble—in a way that they generate more effective results.

This is exemplified by Erbacher's [1] visualization work on the representation of network scans. By narrowing the focus of the approach, the desired activity become more visible. This is the case even with advanced persistent threats such as low and slow scans. This work still has the issue in that the visual display requires visual analytics, interactive exploration, to identify scans. The hypothesis is that incorporating additional approaches to assist the visualization will provide more effective, narrowly focused, visual displays.

Thus, the explored approach is focused on determining if some low-complexity graph algorithms can be used to filter noise and aid detection of sophisticated scans. In particular, to distinguish innocuous traffic from malicious traffic, and to reduce analyst effort in identifying scans. Related activities will manifest themselves in graph connectivity, and some patterns of connectivity such as cliques and strongly connected components reveal ongoing scanning attacks. Semi-automatic clique detection with human interaction was chosen as opposed to fully automatic detection, since determining if a selected group of nodes form a clique is polynomial time, while finding all cliques in a graph is NP-hard. Other graph algorithms that are suitable for real-time detection must all have low-complexity, for instance, finding strongly connected components is linear $O(V + E)$. Clearly, the algorithmic approaches aren't going to fully resolve the analysis process, due to the inability to formalize what is an anomaly; or more appropriately which anomalies are of interest due to their likelihood of being malicious. Thus, identifying cliques and representing the cliques visually for user analysis provides a compromise between excessive analyst time and excessive computational preprocessing.

2. CHARACTERIZING SCAN ATTACKS
There are many types of scans, each with a unique signature. Erbacher [1] provides a more thorough treatment of known scans. The objective of this paper is not to reiterate this work or provide complete coverage of countermeasures against these scanning attacks, but rather to address the common characteristics of all port scans, and demonstrate that visualization of graph features can improve port scan detection, in terms of detection accuracy and detection time.

The goal of port scans, exemplified by `nmap`, is to discover services running on hosts in order to discover the known vulnerabilities associated with those services that can potentially be exploited; this relies on the ability for tools like `nmap` to not only identify what service is running but also what specific software environment and version are providing that service. The ability to locate services relies on the fact that server program typically run on known reserved port numbers, allowing those services to be easily found by any visitor; imagine the chaos of attempting to connect to web sites if every web server used a different port for this service. On the other hand, relocating services to different port numbers is a common defensive strategy for services that should not be open to the public, which is most often done with `ssh`. The attacker can still locate the relocated service based on the connection negotiation with the host on the relocated port but the effort to locate the service becomes challenging and the required scanning will likely become obvious even with sophisticated scan approaches, e.g., asking for an `ssh` service on an invalid port would be a dead away even with only a single instantiation. Thus, an attacker can discover what services are running on a host by identifying what port numbers appear to be active.

The accumulation of results from a network also creates a profile of the network at large, identifying connectivity, service distributions, and also negative space that can be avoided in the future to further improve the undetectability of scans. While not providing a complete treatise of different scan approaches, we will describe several scan approaches from the view of identifying how the attacker can be identified and the appearance of the scan to the network analyst. Clearly, some scans are easy to detect, while others can be quite challenging as some classes of attackers are as concerned with not being detected as they are with locating vulnerabilities and thus employ sophisticated approaches.

2.1 Naive Scans

A naive scan won't try to hide its intention, which makes it easy to detect. For a focused scan, there will be a sequence of packets sent to a host with different port numbers trying to find out what service is running on the host, or a sequence of packets sent to a range of hosts with the same port number trying to discover which host is running the service. A scanner without prior knowledge of which host or which service exist will try to probe everything. The sequence of packets are sent in a rapid fashion, so it can easily be caught within a short window of time.

2.2 Fast Distributed Scans

A naive scan can be easily detected and thwarted. If a firewall is used in the defense of the network, simply blocking all packets from the identified remote host can provide enough protection against naive scans. A sophisticated attacker can avoid easy detection by sending probes from different IP addresses, using spoofed IP addresses or addresses of previously compromised hosts.

If multiple probes are sent from different IP addresses and none of them is on the firewall's blacklist, it will go undetected by the firewall. A distributed scan consists of probes sent to the same local IP address with different port numbers trying to find out what service is available on the host, or probes sent to different local addresses with the same port number trying to find out which computer is running a specific service. A distributed scan completed in a short time can still be detected due to the volume of generated traffic.

2.3 Low and Slow Stealthy Scans

It is low and slow scans that become truly difficult to detect, especially if they also include distribution. This derives from the fact that a *malicious scan* is not defined within any protocol at either the packet or flow level. Network managers will often initiate limited scans to determine if there are any network problems or unauthorized services. Additionally, individual packets can just as easily be part of normal network traffic as part of a network scan; it is the correlation of activity that generally identifies a network scan. Thus, it can be difficult to conclude that a packet is a malicious scan just by using information from one packet or one connection. An innocuous packet sent by a legitimate user can occasionally use a wrong IP address, or a wrong port number. In order to distinguish a packet of a low and slow scan from an innocuous packet, we have to use connectivity information from a long time window.

Stealth scans are designed to remain invisible for as long as possible. Using distributed, low and slow attacks without generating large network traffic can be hard to detect if the observation is made only within a small time window. However in order for the remote attacker to hunt for specific data or system objectives, the collected intelligence must be transmitted back to the remote control and the information together must reveal enough useful information to allow further exploits. The pattern of the scan is to gradually collect complete information in order to conclude which host is running what service while trying to avoid detection as much as possible. The scan pattern is in fact predictable and can be used for early detection.

3. PRIMARY GRAPH MODEL

As implied, the task of identifying scans essentially requires the correlation of individual network packets and connections, i.e., flows. We can model the relationship among flows as an undirected graph $G = (V, E)$, where each vertex $v \in V$ represents a single flow or connection. We can use preprocessing techniques to filter away the flows with normal traffic data, and only show the flows suspected of a scan. Such flows include those that have only one packet, or a TCP connection without payload data, etc. A set of rules can be used to filter away most of the normal traffic, which is considered noise for scanning attack detection.

Each vertex $v \in V$ is a unique 3-tuple that consists of *(Rip, ip, port)*, where *Rip* is the remote host IP address, *ip* and *port* are the local host IP address and port number. Two vertices are connected by an edge $e \in E$ if they have the same *Rip*, *ip* or *port*. Using this graph model, connections from the same remote host will form a clique; connections targeted at the same local host will form a clique; and connections targeted at the same local port number will also form a clique in G. An additional filter can be applied to disallow edges between two vertices that have the same *(ip,port)* combination. If multiple flows share the same *(ip,port)*, it is most likely due to misconfiguration at the local host that causes innocuous connections to be unsuccessful.

On this graph, an innocuous flow with an unsuccessful attempt will appear as an isolated vertex with no connectivity to others. A naive scan from the same remote host will form a large clique in a short time window. A distributed scan from a different remote host targeted at the same local ip address trying to find out what service is available on the host or targeted at the same port number trying to find out which computer is running a specific service can also form a clique. A low and slow stealthy scan will initially appear as isolated vertices in order to avoid detection, but eventually when the missing pieces to the puzzle are collected, there will be a large strongly connected component in the graph.

With the primary graph, the simple distributed scan that directly scans all port numbers or sweeps all IP addresses can easily be detected. The only scan that cannot be easily observed is the low and slow scan that comes from different remote hosts and targets different local IP addresses and/or local port numbers. Scanning very slowly without generating a large volume of network traffic is a stealth technique. It must be observed for a long period to get the big picture.

4. SECONDARY GRAPH MODELS

Typically, when a stealthy attacker moves from one compromised host to the next using compromised hosts to send probes, the two remote hosts are located on the same network. Otherwise, if all compromised hosts come from different networks, it will significantly increase the effort of the attacker to compromise many different sites. In this case, the adjacency graph of the remote hosts can help to identify if there is any relation between the individual scans. We call such a graph the secondary graph.

In the secondary graph $G_2 = (V, E_2)$, the vertex set V has the same vertices as the primary graph, but edges are used to represent the adjacency of remote hosts. If two remote hosts are located on the same network, there is an edge between the two vertices. The adjacency of remote hosts reveals the difference between innocuous packets and packets from a stealthy scan— Innocuous packets will show no connectivity on both the primary graph and the secondary graph; but packets from a stealthy scan will show strong connectivity on the secondary graph. Vertices that are disconnected on the main graph but show strong connectivity on the secondary graph is an indicator of an ongoing stealthy scan. This feature can be used for early detection of a stealthy scan before the primary graph raises an alarm.

5. FAST FEATURE EXTRACTION

The graph features we examine include cliques and strongly connected components. These features need to be updated as a graph is growing with time. Efficient algorithms must be designed to allow real-time computation of these graph features. The goal is to minimize human interaction while avoiding high computational complexity, and find a good balance between the two contradicting objectives.

5.1 Cliques

To compute all cliques in a given graph $G = (V, E)$ takes exponential time in $|V|$. It is a well-known NP-hard problem. To work around the high complexity, we can try both an incremental and a semi-automatic approach.

The incremental approach is effective since the vertices are added to the graph one at a time. We can maintain a list of cliques for the current graph and update the list after a vertex is added. We only need to maintain a list of maximal cliques, not *all* cliques. A maximal clique is a clique whose vertices are not a proper subset of another clique. For example, if vertices u, v, and w form a clique, we would keep the maximal clique formed by $\{u, v, w\}$ only, but not the three smaller cliques formed by $\{u, v\}$, $\{v, w\}$, and $\{u, w\}$. Although asymptotically the number of cliques on the list is still exponential in $|V|$, the actual number can be much smaller.

When a new vertex v is added to a graph $G = (V, E)$, it takes linear time in $|V|$ to add new edges. Suppose the node degree of v is d, i.e., $\delta(v) = d$, then it takes only exponential time in d to update all cliques, since all we need to do is to find all cliques in the subgraph induced by the d vertices. The computational time is exponential in d instead of $|V|$. If a subset of the d vertices is a clique, then including vertex v will form a larger clique that will replace the existing clique.

If $|V| \gg d$, it can speed up the computation significantly. Although the total number of cliques in the graph is still exponential in $|V|$, adding a new vertex only needs to update $O(2^d)$ cliques by the incremental algorithm. Even if $|V|$ is not significantly larger than d, this incremental approach is still highly efficient: (1) if $|V|$ is not significantly larger than d every time a new vertex is added, then the existing graph must be very dense, so there will be a small number of large-sized maximal cliques; (2) if $|V|$ is not significantly larger than d only occasionally, then the computational time will be high every time it happens, but it does not happen very often. Overall the incremental approach is still more advantageous than the one that starts from scratch.

In additional to the incremental algorithm, we adopt a semi-automatic approach, which allows a network analyst to specify which group of vertices are interested, and then algorithmically check whether the vertices form a clique. It takes polynomial time in terms of the number of vertices selected to check whether they form a clique. It is relatively easy for human eyes to detect dense connectivity among a group of vertices, but hard for human eyes to check whether they form a clique. By using automatic algorithm detection and human interaction, the detection of cliques can be done in an efficient manner.

5.2 Strongly Connected Components

To compute the strongly connected components of a given graph is only linear time by using depth first search (DFS). However, since the vertices are added one at a time, an incremental approach can further reduce the running time if we only consider the incremental changes induced by the new vertex. In particular, when a new vertex v is added, we only need to check which components the neighbors of v belong to. If all neighbors of v belong to the same component, then only this component is updated by adding v as its member, and other components remain untouched; if neighbors of v belong to different components, then these components will be connected through v to form a bigger strongly connected component. If we keep the partition current, then it only takes $O(d)$ time to update the partition in the worst case, where $d = \delta(v)$, and the amortized cost is $O(1)$ per operation, as opposed to $O(V + E)$ for a non-incremental approach.

6. VISUALIZING THE GRAPHS

Note that by using the primary graph model, two vertices connected by an edge must share one of the three values in *(Rip, ip, port)*. A node's degree can grow arbitrarily large, but the neighbors can form at most three independent sets, i.e., the cardinality of the maximum independent set from node v's neighbors is at most three. If we visualize node v's neighbors as three groups, the group that shares the same *Rip*, the group that shares the same *ip*, and the group that shares the same *port*, we can clearly observe three cliques. The three cliques may be disjoint when node v is removed or have sparse connectivity on some nodes.

The node with the largest degree usually is a node that has high degree in *Rip*, which indicates a naive scan, or high degree in *ip*, which scans all port numbers to find which service is available on that IP address, or high degree in *port*, which sweeps all computers to find which one is running a specific service. These nodes with high degrees either reveal the intention of the attacker or reveal the source IP address of the attacker, and therefore should generate an alert. Based on these observations, the following rules are used to visualize the primary graph:

1. *(Rip, ip, port)* are used as the *x, y, z* axes for 3-D visualization; so the position of a node v is uniquely defined by the three coordinates $(Rip_v, ip_v, port_v)$.

2. the color of a node carries aging information.

3. neighbors of node v are arranged in three perpendicular planes crossing the center defined by $(Rip_v, ip_v, port_v)$: nodes that share the same *Rip* will be put on the plane perpendicular to the X-axis, and so on.

For the secondary graph, vertices with *Rip* from the same network will form a clique, therefore it is convenient to show them in close vicinity. A 2-D graph is sufficient for the secondary graph. There will be disjoint cliques on the secondary graph, which reveals the sites that have been compromised by the same attacker, or the attackers' home networks if the attackers are not coordinated.

7. EXAMPLES

In [1], scans are detected by observing a fan effect at some remote host, local host, or local port number. We use the same examples in this paper to show how they are detected by visualizing graph features. Figure 1 shows a naive scan, and Figure 2 shows a stealthy scan that sends one packet every 5 minutes. Since all packets are from the same remote host, it can be identified as soon as the clique renders.

8. CONCLUSION

This research shows that there are graph theoretical features of scanning attacks that can be used for detection. Graph models were developed to show the relationship of individual scans. Fast online algorithms are proposed to extract the graph features. To achieve automatic detection as much as possible while avoiding solving intractable problems, algorithmic detection and human interaction are integrated to balance computational complexity and the level of automation. The results show that the proposed feature extraction and visualization can effectively detect naive scans, fast distributed scans, and the low and slow stealthy scans. Future work will consider online learning algorithm that can detect the high level signature of sophisticated scanning attacks such as large scale distributed stealthy scans.

	timestamp	Rip	Rport	ip	port
1	8:43:40:849	10.100.1.165	1281	10.7.1.2	4230
2	8:43:40:850	10.100.1.165	1282	10.7.1.2	4231
3	8:43:40:852	10.100.1.165	1283	10.7.1.2	4232
4	8:43:40:853	10.100.1.165	1284	10.7.1.2	4233
5	8:43:40:855	10.100.1.165	1285	10.7.1.2	4234
6	8:43:40:857	10.100.1.165	1286	10.7.1.2	4235

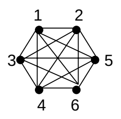

Figure 1: A naive scan targeted at a local host is caught in a narrow period of time.

	timestamp	Rip	Rport	ip	port
1	11:11:30:155	129.123.7.18	31750	129.123.7.80	54
2	11:16:30:177	129.123.7.18	31749	129.123.7.80	55
3	11:21:30:198	129.123.7.18	31748	129.123.7.80	56
4	11:26:30:220	129.123.7.18	31747	129.123.7.80	57
5	11:31:30:242	129.123.7.18	31746	129.123.7.80	58
6	11:36:30:264	129.123.7.18	31745	129.123.7.80	59
7	11:41:30:286	129.123.7.18	31744	129.123.7.80	60

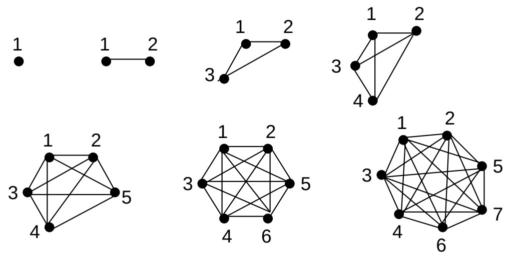

Figure 2: A low and slow scan evolved with time.

9. ACKNOWLEDGMENT

This work was conducted in coordination with the Army Research Lab under the auspices of ARL's summer faculty research program, which is managed by ORAU.

10. REFERENCES

[1] R. F. Erbacher and K. A. Forcht, "Combining visualization and interaction for scalable detection of anomalies in network data," *The Journal of Computer Information Systems*, vol. 50, no. 4, pp. 117–126, 2010.

[2] J. P. Anderson, "Computer Security Threat Monitoring and Surveillance," James P. Anderson Co., Fort Washington, PA (Apr. 1980).

[3] D. E. Denning, "An Intrusion?Detection Model," *IEEE Trans. on Software Eng.*, Vol. SE-?13, No. 2, Feb. 1987, pp 222-?232; also in Proc. of the 1986 Symp. on Security and Privacy, IEEE Computer Society, April 1986, pp 118?-131.

[4] B. Shah and B. H. Trivedi, "Artificial Neural Network based Intrusion Detection System: A Survey," *International Journal of Computer Applications*, Vol. 39, No. 6, pp. 13–18, 2012.

[5] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Trans. Inf. Syst. Secur.*, Vol. 3, No. 3, pp. 186–205, August 2010.