Visualizing Attack Graphs, Reachability, and Trust Relationships with NAVIGATOR *

Matthew Chu, Kyle Ingols, Richard Lippmann, Seth Webster, Stephen Boyer
MIT Lincoln Laboratory
244 Wood St
Lexington, MA 02420-9108
{mchu, kwi, lippmann, swebster, boyer}@ll.mit.edu

ABSTRACT

A new tool named NAVIGATOR (Network Asset VIsualization: Graphs, ATtacks, Operational Recommendations) adds significant capabilities to earlier work in attack graph visualization. Using NAVIGATOR, users can visualize the effect of server-side, client-side, credential-based, and trust-based attacks. By varying the attacker model, NAVIGATOR can show the current state of the network as well as hypothetical future situations, allowing for advance planning. Furthermore, NAVIGATOR explicitly shows network topology, infrastructure devices, and host-level data while still conveying situational awareness of the network as a whole. This tool is implemented in Java and uses an existing C++ engine for reachability and attack graph calculations.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

General Terms

Security

Keywords

attack graph, visualization, treemap, client-side vulnerability, attack path

1. INTRODUCTION

Attack graphs have been proposed by many researchers as a way to identify critical network weaknesses, construct adversary models, analyze network security, and suggest changes to improve security. Over the past few years we have developed a tool called NetSPA (Network Security Planning Architecture) that uses attack graphs to model adversaries and the effect of simple countermeasures. NetSPA discovers all hosts that can be compromised by an attacker starting from one or more initial locations. We previously built a Graphical User Interface (GUI) called GARNET [16] that built on top of the NetSPA system and transformed the output into a more human readable form. We learned many lessons during the design of GARNET and have become aware of several shortcomings that we sought to remedy.

This paper introduces a tool called NAVIGATOR (Network Asset VIsualization: Graphs, ATtacks, Operational Recommendations). NAVIGATOR was designed so users could easily get answers to several key questions, given a chosen attacker model:

1. What machines are at risk?
2. What asset values have been assigned to those machines?
3. What types of attacks are successful?
4. What infrastructure might let attacks through?
5. What needs to be done to prevent attacks?

NAVIGATOR answers the questions above through a series of new features and enhancements to previous attack graph work. To answer the first question, NAVIGATOR shows the types of compromise that can be achieved by an attacker and overlays this onto the network map. The second question is answered by scaling the size of the network elements to represent their relative asset values. These are values assigned to each host before running analysis that represents the utility of hosts to a network's purpose or mission. NAVIGATOR answers the third question by differentiating between server-side, client-side, credential-based, and trust-based attacks. Each of these attack vectors represents a different threat model and administrators need to know what type of attack is used in order to choose the correct remediation. To answer the fourth question, NAVIGATOR explicitly shows the network's infrastructure devices and overlays the attack steps onto the network map. For the last question, NAVIGATOR supplies a set of recommendations and also offers the user the ability to do "what-if" experiments to see the impact of applying these recommendations. Users can also

*This work is sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

VizSEC '10 , September 14, 2010, Ottawa, Ontario, Canada.

tap into all of the information that went into building the attack graph, such as reachability and host-level data.

The rest of the paper describes NAVIGATOR in detail. Section 2 describes our previous work in the area of attack graphs and visualization. An overview of NAVIGATOR is given in Section 3. Section 4 details the enhancements we have made in order to better visualize information the prior GARNET system showed. Section 5 covers the new features that have been implemented in NAVIGATOR. The backend changes that went into the system are described in Section 6. This is followed by a discussion of user evaluations in Section 7, related work in Section 8, future work in Section 9, and a conclusion in Section 10.

2. BACKGROUND

2.1 NetSPA

NAVIGATOR's backend engine is NetSPA, the Network Security Planning Architecture [11, 10, 9]. NetSPA transforms raw data to network models, attack graphs, and other analysis products. It's designed to be fast, scalable, and compatible with common data sources.

NetSPA models both hosts and network infrastructure devices such as firewalls and routers. It assumes that hosts can have one or more open ports that accept connections from other hosts and that ports have zero or more vulnerabilities that may be exploitable by an attacker. Individual vulnerabilities provide one of four access levels on a host: "root" or administrator access, "user" or guest access, "DoS" or denial-of-service, or "other," indicating a loss of confidentiality and/or integrity. Vulnerabilities can either be exploited remotely from a different host or locally from the vulnerable host. Currently, it is assumed that an attacker obtains a host's reachability if root- or user-level access is achieved.

NetSPA's importer component reads in raw data such as Nessus scans, firewall rulesets, and National Vulnerability Database (NVD) records [1], forms the data into a network model, and converts the data into a custom binary file format. The main computation engine, written in C++, is responsible for reading in the binary file, computing reachability, generating attack graphs, analyzing the graphs to generate recommendations, and computing security metrics. The computation engine is connected to the Java-based viewer component via Java Native Interface (JNI).

NetSPA's reachability system emulates the network's firewalls, computing reachability between hosts in the network and between *locations*, such as the Internet-facing network segment, and the rest of the network. The latter capability allows NetSPA to posit an attacker with any IP address, permitting a worst-case analysis that can discover interesting holes in firewall rulesets that often cannot be discovered by scanning.

Originally, NetSPA only supported server-side attacks, handling cases where an adversary contacts a vulnerable server (e.g., a web server) and effects compromise. Recently, NetSPA has been extended to model "client-side" attacks, in which the victim contacts the attacker (e.g., a vulnerable web client contacting a malicious server). To support client-side attacks, NetSPA additionally computes reachability *backwards*. NetSPA's reverse reachability system allows for worst-case analysis, determining which server addresses could effect compromise.

Reachability is time-consuming to calculate and could easily dominate the computation time of the system. NetSPA takes steps to minimize the number of calculations that must be performed by identifying potentially redundant work before it's performed. Prior to reachability computation, NetSPA identifies hosts with identical reachability and collects them into *reachability groups*. In the worst case this offers no savings, but for networks seen in practice the savings are substantial.

On top of the reachability system, NetSPA builds network-based *attack graphs*. NetSPA has a graph structure called the multiple-prerequisite (MP) graph [10], a time- and space-efficient graph that represents the entirety of the modeled attacker's potential actions. An attack graph shows, for a given attacker starting location or locations, the potential steps the attacker could take through the network. Each of these steps has an attack graph depth associated with it, which represents how far the attacker has progressed. For instance, a machine at depth 1 can be directly compromised by the attacker, whereas a machine at depth 2 requires the attacker to first compromise a depth 1 stepping stone.

2.2 GARNET

An earlier NetSPA-based GUI called GARNET [16] enabled users to explore the data generated by NetSPA, compute metrics that assess attacker effort, and perform experiments that explore defensive measures and adversary types. We learned a great deal from the construction of GARNET and have carried those lessons forward into the NAVIGATOR project.

Many of GARNET's key features have been kept intact. For example, the new NAVIGATOR system not only includes the ability to evaluate the current state of the network, but also the ability to perform "what-if" experiments. These "what-if" experiments allow administrators to understand the impact of applying recommendations to remediate sets of vulnerabilities or introducing new zero-day vulnerabilities.

Although GARNET was a step forward for attack graph visualization, some of GARNET's features needed to be reworked or extended in order to accommodate both lessons learned from the GARNET tool and new features added to the NetSPA engine. The remainder of the paper will explore these changes. Some of the changes are "backend" changes, most notably a modification to the actual graph structure generated by NetSPA, in order to more easily support visualization design. The new graph structure is covered in Section 6.1.

Many of the changes are to the "front-end" of the visualization itself. GARNET did not offer resolution down to the host level, staying at the level of so-called host groups. This is a useful abstraction that is kept; Section 4.1 covers NAVIGATOR's strategy for grouping hosts. However, NAVIGATOR also allows zooming down to far greater levels of detail. Section 5.3 covers the way NAVIGATOR offers

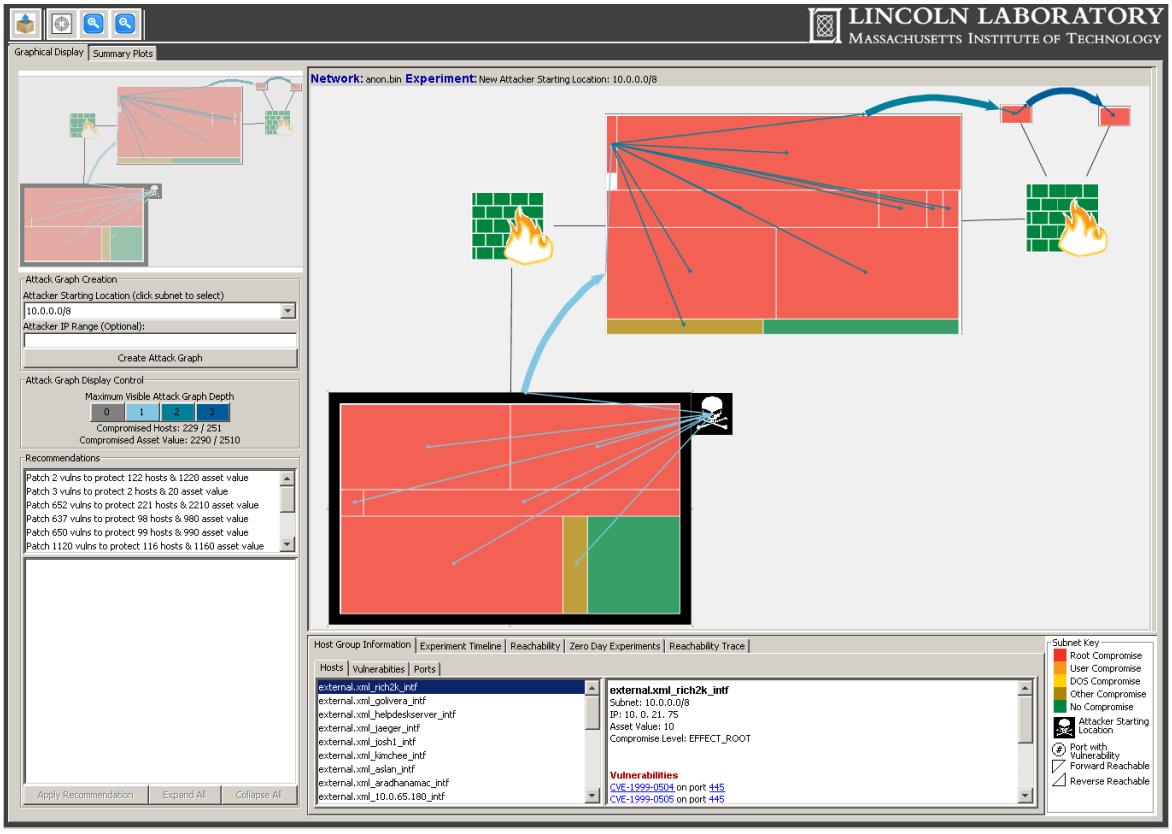

Figure 1: The NAVIGATOR GUI

detail via graduated zooming.

GARNET did not show networks' infrastructure devices. NAVIGATOR makes this explicit. The issues surrounding the depiction of infrastructure are discussed in Section 5.2.

Lastly, GARNET was geared toward server-side attacks only. NetSPA is now capable of modeling client-side threats and credential- and trust-based threats (e.g., the compromise of a Windows domain controller). The NAVIGATOR system has been extended to handle these cases. Visualization of the more complicated reachability in this model is covered in Section 4.2 and the attacks themselves are reviewed in Section 5.1.

3. NAVIGATOR

NAVIGATOR's attacker model expands on GARNET's by giving attackers access to client-side, credential-based, and trust-based attacks. As in GARNET, there are three attacker types: *simple* (has exploits for all known server-side and client-side vulnerabilities and can use credentials and trust relationships), *single-zero-day* (has all of the simple adversary's capabilities and one zero-day exploit for a non-public vulnerability), *comprehensive-zero-day* (has all of the simple adversary's capabilities and a zero-day exploit for every open port on the network). The user can model all of these attackers starting from inside or outside of the network.

Figure 1 shows an example screenshot of NAVIGATOR. The graphical display is made up of three parts: the main screen (center), the control panel (left side), and the information panel (bottom).

In Figure 1's main screen, the attack graph data is overlaid on the network map. The rectangles are subnets and the icons are infrastructure devices. The background colors represent the compromise levels achievable in this attack scenario. The levels, as defined in Section 2.1, are shown as follows: red is root, orange is user, yellow is other, gold is DOS, and green is no compromise. The arrows show the steps that the attacker could take to progress through the network. The color of these arrows show the attack graph depth at which each attack is taking place. The attacker's starting subnet is highlighted by an additional surrounding black box. There is also an attacker icon attached to the upper right of this subnet, which serves as the source for all arrows representing direct compromise from the attacker.

The control panel occupies the left side of the screen and is made of several parts, which we describe from top to bottom. The overview panel always shows the entire network and allows the user to control the main screen's zoom level and position within the network. Next is the attack graph creation panel where the user selects the desired attacker model (starting location) and the depth of attack to be viewed. The attacker's starting location is chosen by clicking a subnet or choosing from the dropdown menu. A user can control the depth of the attack graph shown using the buttons. This

allows the user to see the attacker's progress only up to the selected depth, with all compromise colors being updated and attack graph edges being hidden as necessary. Last is the recommendation panel. NAVIGATOR provides a list of possible improvements to the network's security, each of which provides a set of vulnerabilities to remediate and a description of the sections of the network protected by doing so. The user can also get information about the selected recommendation and perform an experiment by applying the selected recommendation.

The information panel, located at the bottom of the window, consists of several widgets that allow the user to access a wide variety of information and perform experiments. It currently has the following tabs:

- **Host Group Information** - This shows host-level and port-level data about the host group currently selected in the main screen. It has three tabs that give descriptions about the hosts, vulnerabilities, and ports that are present in the selected set. There are links from each one of these tabs to the other two, so the user can easily drill down for more details.
- **Experiment Timeline** - A user performs a "what-if" experiment by either applying a recommendation or adding a zero-day vulnerability. This panel allows the user to move back and forth between the various experiments that have been performed. It is the same as GARNET's timeline, described in Section 5 of [16].
- **Reachability** - By selecting a set of source hosts or host groups using the main screen, users can graphically see all of the host groups that the sources can reach (forward/server-side reachability) or can reach the sources (reverse/client-side reachability).
- **Zero Day Experiments** - This allows users to introduce zero day vulnerabilities into the network and see the resulting impact on network security. By doing this, a user can model different attackers and proactively determine defenses for potential future threats. In this panel, it is also possible to calculate the impact in terms of additional asset value compromised for each potential zero day vulnerability.
- **Reachability Trace** - The reachability trace feature allows users to discover *how* the attacker can reach a given destination port, which is chosen from a drop-down menu of all the open ports on the selected host. The tab shows the hypothetical starting packet information and the specific firewall rules that would allow the traffic to progress.

Using the overall tabs at the top left of the screen, it is also possible to switch from the Graphical Display to Summary Plots. This is similar to GARNET's Summary Plots mode and displays three security metrics that graph the percentage of network assets the attacker captures as a function of different measures of attacker effort. It is possible for the user to compare these metrics across experiments, which further helps to assess the impact of applying recommendations or introducing zero day vulnerabilities. The three metrics are Assets Captured vs. Number of Attacker Hops, Assets Captured vs. Number of Exploits Used, and Assets Captured vs. Cost of Attack.

The first two metrics are described in [16]. The third metric measures the cost of capturing network assets in terms of the sophistication of the exploits needed. This metric assumes that exploits for different types of vulnerabilities cost the attacker different amounts to either create or buy. Exploits for vulnerabilities that have already had patches released are close to free. Exploits for known vulnerabilities without patches cost a small amount. Zero-day vulnerabilities are fairly expensive. This metric gives the user some idea of how far into the network various types of attackers should be able to reach.

4. ENHANCEMENTS

When building NAVIGATOR, the first thing we set out to do was directly apply the lessons learned from GARNET. By doing so, we have been able to substantially enhance the visualization of information GARNET already displayed.

4.1 Host Group Visualization

Users often want to get the big picture view of their network's security. For this task, seeing individual hosts can often end up being overwhelming, so we help the user by grouping similar hosts. *Host groups* are made up of hosts that are all in the same subnet, can all be compromised to the same extent at the same depth, and are treated identically by all network filtering devices. Creating groups in this manner allows us to continue to convey the interesting attributes of reachability and vulnerability.

Furthermore, NetSPA allows the user to specify an *asset value* for each host in the network in order to indicate its relative importance. GARNET did not utilize the information when creating the visualization and instead made the size of both subnets and host groups proportional to the number of hosts. NAVIGATOR scales sizes by the total asset value, simultaneously showing reachability, vulnerability, and value to the user.

In order to lay out the host groups, GARNET uses the strip treemap algorithm [5], which processes host groups in order and puts them in horizontal strips of varying thickness. This allows it to use 100% of the available space and generally creates good aspect ratios for the laid out rectangles. However, it gives no guarantees about the aspect ratios and when constructing attack graphs, it is not uncommon to see situations that lead to extreme aspect ratios (ie the rectangles are either very long and thin or very short and wide). Such rectangles are both æsthetically unappealing and difficult for the user to interact with, e.g., by clicking. The small rectangles in Figure 2 are examples of this type of rectangle.

The introduction of zooming (described in Section 5.3) also exacerbates the extreme aspect ratio problem. When switching from showing host groups to showing hosts, it is jarring to the user if the hosts do not stay "in place", which means that the hosts must occupy the rectangle formerly representing the host group. If the host group's rectangle has an extreme aspect ratio, the host rectangles we put inside it will necessarily continue to have extreme aspect ratios.

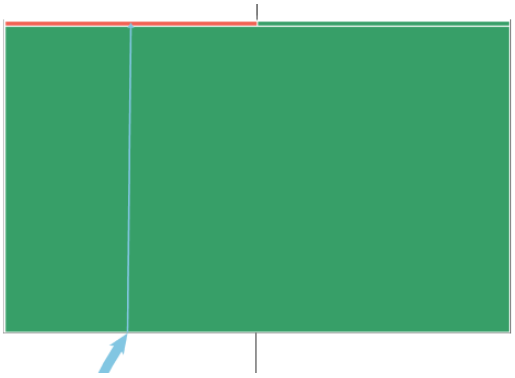

Figure 2: Subnet layout using unmodified strip treemap algorithm. The compromised host group (top left) is very thin and difficult to interact with.

We considered a range of solutions to this problem and judged them based on their ability to also satisfy the following criteria.

- **Handle multiple asset values** - Because asset value is being shown by size, the relative size of each rectangle is important. Host groups generally do not have the same asset value, and the corresponding size difference must be maintained.
- **Only a small amount of wasted space** - Both the size of the subnet and the size of the host groups are significant. This means that any wasted space must cause slight deviations in at least one of the two, creating slight inconsistencies with the rest of the network. Some wasted space is acceptable and expected, but it must be minimal.
- **Rectangular shapes** - After size and color, the shape of the host group is its most visible attribute. We were worried that if host groups took on non-rectangular shapes, users would begin to associate meaning to the shapes themselves. Therefore, shape is not a good free variable for the layout system.
- **Maintain order** - For host-level zoom (Section 5.3), the hosts are laid out in sorted order. Thus, maintaining this order at the group level helps the user more quickly find a particular host of interest.

Our solution is to modify the strip treemap algorithm to ensure every rectangle had a minimum dimension for both width and height. Comparing Figure 2 to Figure 3, one can see the effect of our algorithm. At the small expense of some wasted space, both of the top rectangles have been made significantly easier to see and interact with. Here is the strip treemap algorithm with our additional modifications in bold.

Input: A *layout rectangle* to be subdivided and a set of *input rectangles* that are ordered and have given areas

1. Scale the area of all the rectangles so that the total area of the input rectangles equals that of the layout rectangle. **Define the minimum height (M_h) and the minimum width (M_w).**[1]

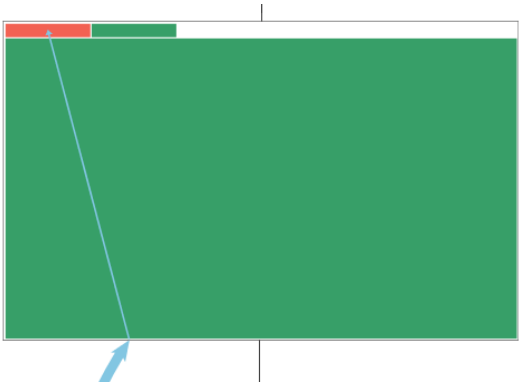

Figure 3: Subnet layout using modified strip treemap algorithm. The compromised host group is now easier to view and select, at the expense of some wasted space.

2. Create a new empty strip, the current strip.
3. Add the next rectangle to the current strip, recomputing the height of the strip based on the area of all the rectangles within the strip, and then recomputing the width of each rectangle. **If the calculated height is less than M_h increase the height to the M_h and decrease the width so that input rectangle's area remains unchanged. Do the same for the width using M_w. Keep track of the total extra height added and the maximum extra width from any row.**[2]
4. If the average aspect ratio of the current strip has increased as a result of adding the rectangle in step 3, remove the rectangle pushing it back onto the list of rectangles to process and go to step 2.
5. If all the rectangles have been processed, **go to step 6**. Else, go to step 3.
6. **Let L_h and L_w be the layout rectangle's height and width, respectively. Let E_h and E_w be the total extra height and maximum extra width, respectively. Scale the height of all rectangles by $\frac{L_h}{L_h+E_h}$. Scale the width of all rectangles by $\frac{L_w}{L_w+E_w}$.**

[1] M_h and M_w must be less than or equal to the layout rectangle's height and width, respectively, because otherwise the input rectangles would be taller or wider than the layout rectangle. M_h * M_w must be less than or equal to the smallest input rectangle's area because otherwise any rectangle that met both minimum dimension criteria would necessarily be too large. NAVIGATOR uses a default of 25 for both M_h and M_w because in our visualization scheme, this value balances making the rectangles easy to interact with and not wasting too much space.

[2] Since this algorithm is placing the input rectangles into independent rows, the maximum extra width among the rows is also the total extra width that we must deal with for the layout rectangle.

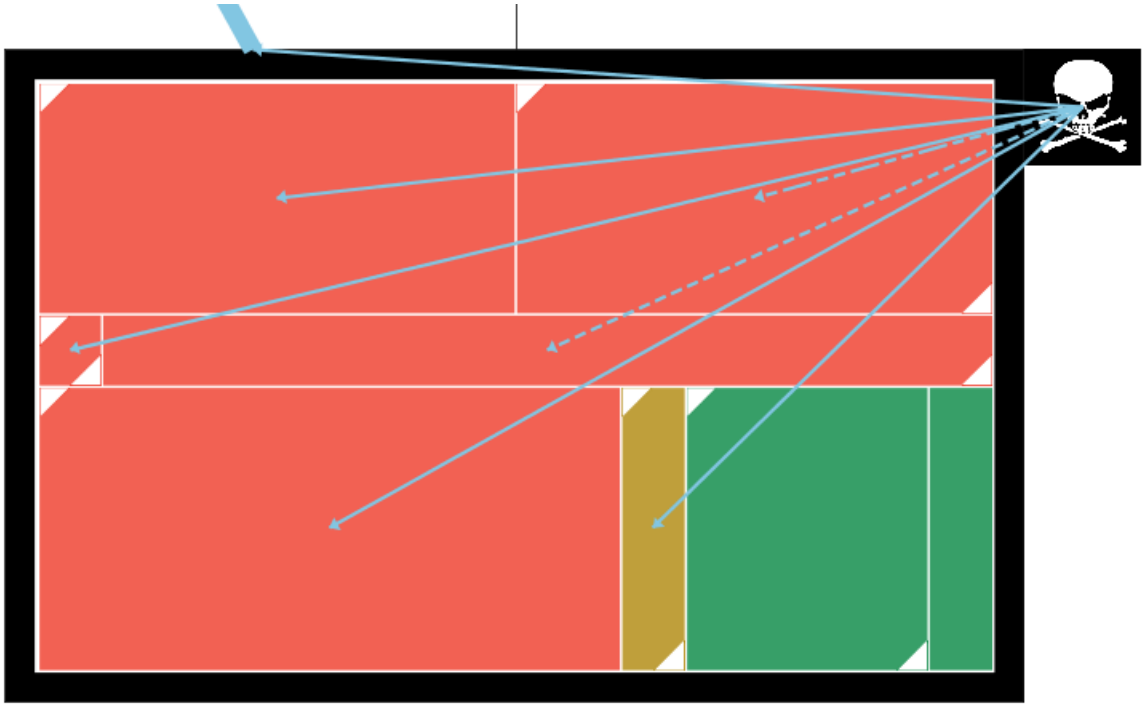

Figure 4: Host groups showing reachability. A white triangle in the upper-left indicates forward reachability to the group; the lower-right indicates reverse reachability from the group.

The purpose of step 6 is to ensure that the input rectangles continue to fit inside the layout rectangle. It can cause the dimensions of rectangles to again dip below the minimum dimension, but it never gets too far below because adding additional height or width is rare. This is because the strip treemap algorithm itself seeks to avoid extreme aspect ratios. The maximum extra width never gets too large because doing so requires lots of tall narrow rectangles, which the algorithm naturally avoids by moving rectangles to the next strip. The total extra height never gets too large because doing so requires lots of short wide rectangles, which the algorithm naturally avoids by moving more rectangles onto the current strip.

4.2 Reachability

Reachability information is a key prerequisite to attack graph generation. It is often of value to users to see it in its own right. This can help the user to verify that all filtering devices in the network are behaving as expected, which is difficult to do by hand given the sheer number of filtering devices and rules in most networks. It can also help an administrator decide how to configure the network by showing the overall connectedness of the network and identifying any reachability bottleneck hosts.

In GARNET's network map mode, the user had the option for every host group to display outgoing and/or incoming reachability. However, there were some problems with its approach. Since reachability links were simply drawn as directed arrows, looking at reachability for multiple host groups often led to a confusing jumble of arrows that was hard to follow. Furthermore, attack graph edges were also drawn as directed arrows and thus could not be shown at the same time as reachability because the user would be unable to differentiate them. It is useful to be able to look at reachability and attack graph edges at the same time because looking at the difference between them helps to highlight potential *latent threats*. For instance, the presence of reachability without an attack graph edge means that the machine is currently safe, but may become susceptible to attacks in the future if and when vulnerabilities are discovered in its software.

NAVIGATOR solves both of these problems by not showing reachability through arrows and instead adding extra symbols to highlight reachable and reverse reachable host groups. This approach allows users to see reachability while examining the network as a whole because even for large networks, the user should be able to pick out these specialized symbols. When actually picking the symbols, we took the following factors into account.

- **Fit Inside 25 by 25 pixel box** - If the size of the symbol varies in different host groups, users may associate meaning to this size. This meant that we needed our symbol to be small enough to fit inside any size host group. Since our layout algorithm allows both the host group's width and height to be as small as 25 pixels, this means the symbol must fit inside a 25 by 25 pixel box.

- **Independent of Compromise Color** - Learning about reachability in NAVIGATOR is decoupled from the attack graph in the sense that it is possible to change the

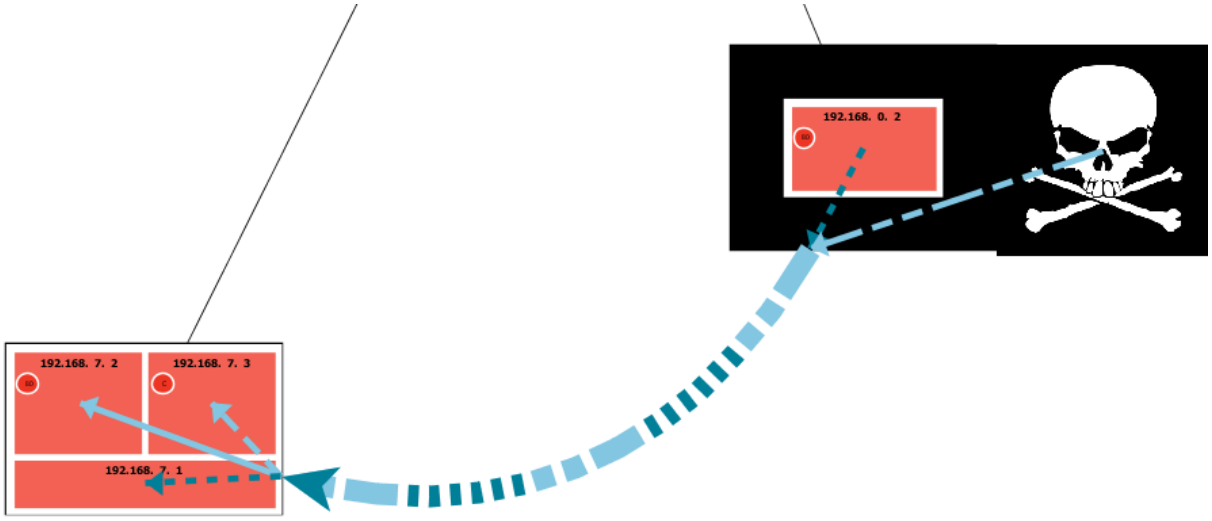

Figure 5: Attack type and depth indicated in a single edge. The attacker can use server-side (solid) and client-side (dashes) at depth 1 (gold) and a trust relationship (dots) at depth 2 (purple).

attack graph without changing the reachability sources (and vice versa). Therefore, we sought to make the visualization of the reachability stable regardless of the host group's color based on compromise level.

If the sources can reach ports on the host group (forward reachability), a white triangle is shown in the host group's upper left corner. If the host group can reach ports on the sources (reverse reachability), a white triangle is shown in the host group's lower right corner. Examples can be seen in Figure 4. The upper left and bottom left host groups are only forward reachable. The middle right host group is only reverse reachable. The bottom right host group is not reachable at all. All others are both forward and reverse reachable. The uncompromised host group that is reachable is an example of a latent threat and it is likely that adding a zero-day vulnerability would cause this host group to become compromised.

The drawback to this approach compared to GARNET's is that the user can only visualize the reachability of one set of sources (a host, host group, or subnet). However, we could not develop any use cases that required the user to simultaneously look at the reachability from multiple sets of sources. In all cases, looking at the reachability of sets of sources in series was sufficient.

Showing reachability via edges is even more intractable when per-host data is shown. At this level, we lose the benefit of host groups and the clutter from lots of edges is even worse. Reachability is shown through symbols on the reachable ports themselves. This serves as an easy way to concisely convey to the user that the host is reachable while also showing port-level information. The user is also able to see a list of the reachable hosts (and reachable ports on that host) in the Reachability tab of the Information panel. This information was not easily accessible in GARNET.

5. NEW FEATURES

In addition to improving the visualization offered by GARNET, NAVIGATOR has many additional features that convey new information.

5.1 Client Side Attacks and Trust Relationships

Client-side attacks are an increasingly common attack vector where attackers exploit vulnerabilities in web browsers, e-mail clients, document viewers, and multimedia applications running on victim machines. In [9], we describe changes to the NetSPA system in order to model client-side attacks, including using an Open Vulnerability Assessment Language (OVAL)-based scanner to detect the client-side vulnerabilities and calculating reachability *backwards* to efficiently determine if a victim's client can reach a malicious server.

Additionally, NetSPA is capable of modeling trust relationships, where certain machines are given privileges to administer other machines. We currently do not have an automated input mechanism for trust relationships, but it is possible to manually input them into the system.

It is important to differentiate between server-side, client-side, and trust relationship attacks in the GUI because their respective countermeasures vary greatly. In order to do this, we set out to differentiate the arrows that correspond to these attacks. The most obvious element of the arrow to change would be its color, but that attribute is already significant because it corresponds to the attack graph depth. Therefore, it was decided that the line type of the arrow would specify the attack type.

Furthermore, NAVIGATOR aimed to create a cleaner display by aggregating all edges between subnets into a single edge. Previously, in GARNET, edges were simply drawn as arcs directly from the source host group to the destination

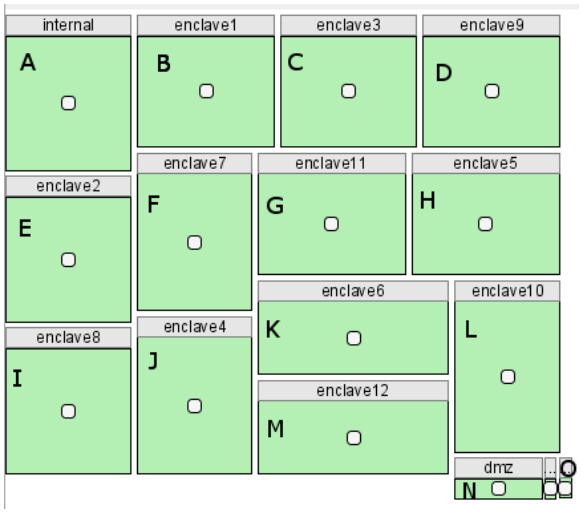

Figure 6: GARNET's network view. Infrastructure devices are absent.

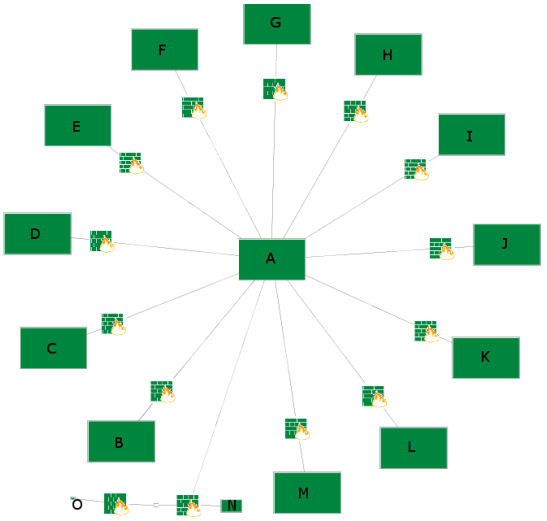

Figure 7: NAVIGATOR's network view, showing infrastructure devices and network topology.

host group. This would often result in a jumble of edges that cluttered the view. By aggregating edges, NAVIGATOR highlights the progression of attackers between subnets, which is often the most critical type of attack.

The new aggregation method requires us to draw all inter-subnet arrows as the combination of three distinct arrows: the actual edge that connects the subnets, the source side edge that connects the source host group with the actual edge, and the target side edge that connects the actual edge to the target host group. However, this leads to problems when there are multiple attacks between the same source and destination that do not share the same attack type or attack graph depth.

We considered creating a separate edge for each (depth, type) pair and stacking these edges adjacent to one another. However, because there is no upper limit to the number of such pairs, a large number of edges could be created, which causes the same cluttered view that we were trying to solve. Furthermore, this would make some subnets have more edges between them than others, which the user could misconstrue as an indication of relative severity.

We avoid these problems by creating hybrid edges that combine multiple colors and multiple line types. The system essentially renders all (depth, type) pairs at once, in a single edge.

Attack types are indicated by solid, dashed, and dotted sections of each edge, representing server-side attacks, client-side attacks, and trust relationship exploits, respectively. When reading in an attack graph, NAVIGATOR maintains a map from a pair of subnets (the source and destination) to the inter-subnet edge that would go between them. Each inter-subnet edge is responsible for keeping track of the various types of attacks it represents and we use the map to find and update the attack types every time an inter-subnet edge is encountered. In this way, NAVIGATOR knows what types of lines need to be included in the arrow representing this particular attacker step.

To handle different attack depths, multiple colors are displayed by creating a different edge for each depth and assigning different colors based on the depth. Then, we configure each individual arrow's rendering so that together they create an overall arrow that cycles between the various colors and maintains the specific line type within each one. This requires adding blank spaces into the arrows and staggering their starts so that they occupy distinct segments of the overall arrow being drawn. Figure 5 shows an example where the attacker is able to use both a server-side and a client-side attack at depth 1 (shown in sky blue) and a trust relationship exploit at depth 2 (shown in teal). Each of these attacks is used to compromise one host on the other subnet.

5.2 Infrastructure

NAVIGATOR shows the infrastructure in the network, a key piece of information that GARNET omits. While this is not a full physical view of the network, it is still very informative to see how the subnets are connected by infrastructure devices and which devices sit along which paths between various subnets. This also helps to identify devices that are serving as the gateway between many subnets and are thus single points of failure should an attacker manage to affect them in some way. Furthermore, when an attack graph is created, it is very important to see which firewalls and routers are being circumvented and are thus not effective in preventing attacks.

NetSPA models infrastructure devices as multihomed hosts with rules dictating the flow of traffic among the interfaces. When reading in the network, NAVIGATOR simply creates different nodes for these devices (separate from the nodes created for subnets) and creates edges for the appropriate connections. It is important to differentiate between subnets and infrastructure devices because they serve different purposes in the network and, as a result, have different meanings to an administrator. Infrastructure devices are drawn using icons.

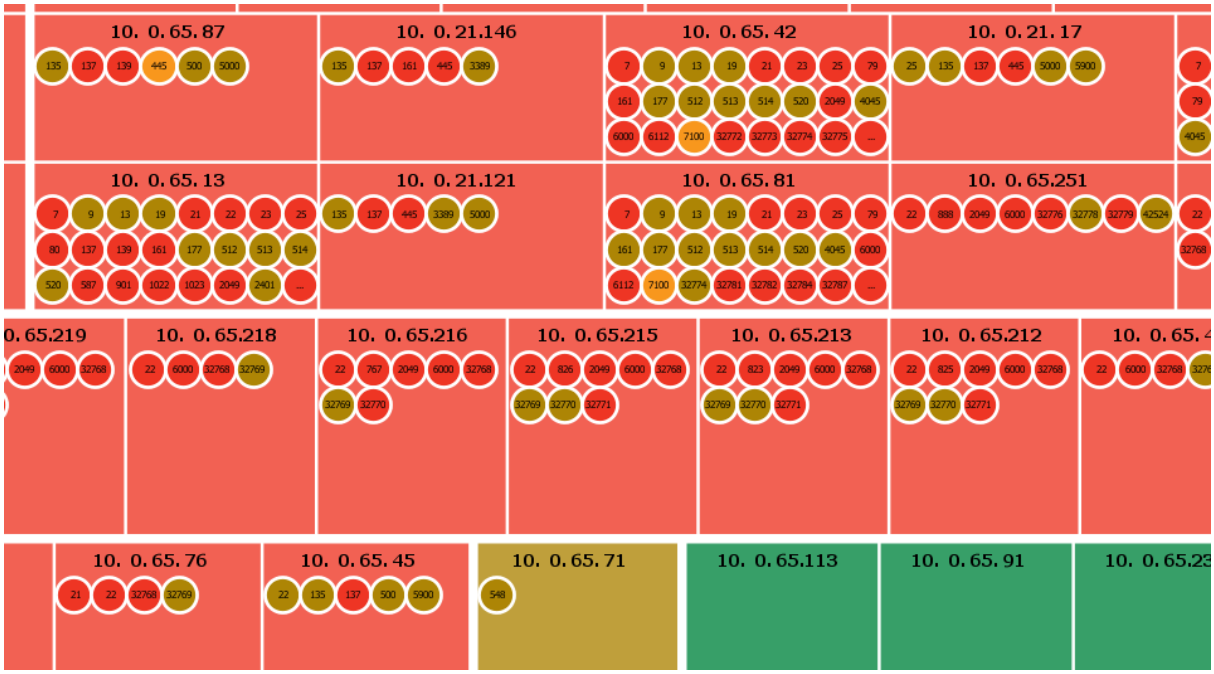

Figure 8: Host-level zoom, showing IP addresses, ports, and port vulnerability.

Because it shows infrastructure devices, NAVIGATOR has two key advantages over GARNET. First, the infrastructure devices' presence in NAVIGATOR's network graph leads to more accurate network maps being drawn because the graph layout algorithm is now aware of all the connections that exist. GARNET simply uses a grid layout for the subnets, ignoring the network's topology entirely. Comparing Figures 6 and 7, one can see that using the infrastructure devices allows the user to get a much clearer picture of what the network looks like. The subnets (labeled with the letters A-O) in Figure 6 roughly correspond to the subnet with the same letter in Figure 7. However, the icons in Figure 7, which represent infrastucture devices, have no corresponding representation in Figure 6. Without them, it is hard to know the topology: in this case, there is a single central subnet that has routers to connect it to each of the enclaves.

Also, NAVIGATOR is able to show when infrastructure devices themselves are compromised by changing the background color of the icon representing that device. GARNET separated the interfaces that comprised the device and put each one in the subnet that it faced. Therefore, when an infrastructure device was compromised, there was no good way to convey this information because there was no way for the user to know that the interfaces were associated with one another. The best choice was showing all the interfaces as compromised, which falsely looked to the user like many different hosts were being compromised.

5.3 Zooming

NAVIGATOR adds more detail as the user zooms in, attempting to always keep a rich quantity of information on the screen. Users may utilize any of the mouse wheel, network overview panel, or zoom in/out buttons to control the level of zoom. They may also use any of the network overview panel, the scroll bars, or drag-and-click to control their position within the network. Instead of constantly making the host groups bigger, there is an established threshold at which NAVIGATOR will begin to show individual hosts within the host group. This makes it possible for the user to access the wealth of information that NetSPA collects about individual hosts. By waiting until the user zooms in far enough, the user is not overwhelmed by all this information and is still able to view the bigger network security picture.

NAVIGATOR graphically displays selected information within the host's rectangle. The laying out of hosts within a host group also utilizes the modified strip treemap algorithm described in 4.1. This turns out to be very important because knowing that there is a minimum dimension for each side of the rectangle helps us to ensure that the details NAVIGATOR displays about each host can be seen. A great deal of additional information about the host (such as open ports and existing vulnerabilities) can be accessed by clicking in the host's rectangle and looking at the Host Information tab.

NAVIGATOR displays the host's IP address at the top center of the host's rectangle. The rest of the rectangle is occupied by circles corresponding to open ports. Since space is limited, we chose to draw only those ports that currently have vulnerabilities. The circles' colors indicate the worst level of compromise that can be achieved by exploiting one of the vulnerabilities on the port, using the same color scheme used for the hosts and the host groups. Clicking on a particular circle will select that port number on the Ports tab on the Host Information tab.

Since there is no limit to the number of ports that can belong to each host and only a limited amount of space in the host's

rectangle, there is a possibility that the number of slots will be less than the number of ports to show. In this case, the last circle shows an ellipsis to indicate that there are more ports to be shown. This circle's color indicates the worst compromise level among the unshown ports.

An example of this can be seen in Figure 8. There are varying numbers of open ports in the hosts shown, including some hosts that needed the ellipsis. There is also variation in the severity with different ports having their worst level of compromise be root, user, or other.

Additionally, once individual hosts are being drawn, the number of attacker arrows shown must be limited because a single arrow drawn to a host group could turn into tens or even hundreds of arrows, depending on the size of the host group. This would severely clutter the screen and obscure the very details that the host level zoom is meant to convey. The compromise we decided on was to only show attacker arrows that crossed between subnets or use credentials or trust relationships. This strikes the correct balance between still showing the important steps of an attacker's progression and not cluttering the screen.

6. BACKEND IMPROVEMENTS

Not all of the changes to the system are visible to the user. Several improvements to the NetSPA backend itself were made to improve the performance of NAVIGATOR, by migrating computation to the C++ engine and reducing the quantity of data that must be handled via JNI interactions between the GUI and the backend.

6.1 MPRGGraph

The "native" attack graph structure produced by the NetSPA engine is the multiple-prerequisite (MP) graph, as described in [10]. For space reasons it is not described here. It is a useful data structure, but it is not easily used by a visualization tool. It is not practical to simply walk the graph in order to determine what is compromised when. Attacks which have *prerequisites*, such as obtaining a password or other credential, are more complicated – the attacker may obtain reachability to the vulnerable port well before or well after obtaining the other prerequisites needed to exploit it. Additionally, because the MP graph encodes attacker access via *state* nodes that combine both level of access and the host involved, a given host may appear in several different graph nodes. This complicates efforts to develop a GUI that uses a network-based visualization, in which hosts appear only once.

The GARNET system performed extensive Java-side computation to utilize the native MP graph structure as-is. The result was a complicated system that had trouble with multi-homed hosts, prerequisites other than reachability, and trust relationships. We have learned from that experience and have implemented a new graph type in C++ for the benefit of the GUI.

The new graph, a multiple-prerequisite reachability group graph or MPRG graph, is derived from both the native MP graph and from reachability information generated by NetSPA's reachability engine.

The MPRG graph has a straightforward relationship between hosts and links on the network and nodes: a given host belongs to exactly one MPRG graph node, no more, no less. Two hosts exist in the same MPRG graph node if they are on the same subnet(s), have identical reachability to other parts of the network, and are compromised at the same depths (e.g., the attacker can directly obtain user-level access but requires one stepping-stone to gain root-level access). Thus, an MPRG node's contents correspond exactly to a host group in NAVIGATOR. With this data the graphical system can efficiently show which hosts are compromised at which depths and to what extent.

The MPRG graph's edges contain additional data. A given edge indicates that the source node is able to compromise the target node. The edge identifies any credentials or trust relationships employed in the attack, the depth of the attack, and the resulting level of access. Note that edges always go from hosts able to actually carry out the attack against the targeted host, regardless of where any credentials employed in the exploit were obtained. (It is possible, for example, to obtain a password from a host X but be unable to reach the vulnerable host Y from it – rather, the adversary requires a different stepping-stone host Z to actually carry out the attack. The MPRG graph shows the attack from Z to Y.)

The MPRG graph is constructed in two basic stages. First, the hosts in the network are grouped into MPRG graph nodes, such that each host belongs to exactly one node. Second, edges are created between nodes to indicate potential attacker activity. Note that not every node will have an edge to it; hosts that do not have known vulnerabilities may exist in this state.

Hosts are grouped based on their compromisability and their reachability. Because the MPRG graph indicates the depth at which a given level of compromise occurs, hosts cannot be grouped unless they are compromised similarly. For example, an MPRG graph node could indicate a group of hosts compromised at the *user* level at depth one (meaning a direct attack) and *root* level at depth three (meaning two stepping-stone attacks were used). Hosts must additionally be in the same *reachability group*, meaning they have identical reachability to and from other devices in the network.

Edges between nodes represent potential attacker actions and are therefore directional. Each edge contains the depth at which the edge's action ends and the identity of any credentials or trust relationships used in the attack. The depth at the target may not be simply the depth at the source plus one; an exploit that requires a credential collected elsewhere can nevertheless be executed from an earlier stepping-stone with reachability to the target. Note also that a pair of MPRG graph nodes could have more than one edge connecting them, e.g., if there is both a credential-based attack and a "plain" attack possible on the same host.

In our implementation, nodes are formed by first computing a hash value for each host based on its compromise and reachability attributes and then grouping based on identical hash value.

Edges are far more complicated. The straightforward ap-

proach is to simply explore every compromise represented in the underlying multiple-prerequisite graph, by starting at each state node in turn and walking "down" the graph to all reachable states. However, this is essentially quadratic in the number of hosts. Instead, we explore every *reachability group*, walking backwards to discover the hosts using the group and forwards to discover the hosts exploited and vulnerabilities used through the group. These walks give rise to one or more edges, grouped based on the credentials used to effect the compromise and on the nodes representing the perpetrating and victim hosts. Because exploration is group-centered – exploring each reachability group once, rather than once per pair of attacker and victim using the group – we obtain a substantial time savings over the straightforward approach.

6.2 Speed

One of GARNET's shortcomings is that it is very slow on the two most data intensive actions of the system: loading a new network and viewing security metrics. For instance, loading a network of 20,000 hosts spread over 100 subnets takes less than a second in NAVIGATOR, but over a minute and a half in GARNET. This is because GARNET preloads all data that it could potentially use over the course of the user's interaction. Some of this data does not need to be loaded immediately and often doesn't even get used at all. Specifically, all of the reachability computations do not need to be calculated when the network is loaded and the data for all three metrics do not need to be loaded when the user switches to the metrics tab. NAVIGATOR fixes this problem by waiting until the data is actually needed before asking the engine to calculate it. For the reachability calculations and two of the three metrics, the engine is fast enough that the user cannot distinguish between when the data is preloaded and when it was loaded on demand. The third metric (assets captured vs. number of exploits used) is the result of many random experiments and cannot be loaded instantaneously. Therefore, instead of forcing the user to wait, the chart is updated after each experiment and the results up to that point are made visible to the user.

7. USER EVALUATIONS

There were many driving forces behind the decision to redesign NetSPA's GUI. Discussions with network defenders who were users of the tool identified the need to include infrastructure devices and network topology, which were both missing from GARNET. Improvements to the NetSPA engine made calculating client-side attacks and trust relationship exploits possible, which in turn made visualizing them possible. However, the main force was a process of internal evaluation that started with the GARNET system itself. As described in [16], GARNET was evaluated by several users with a technique called heuristic evaluation. The unresolved comments from that evaluation formed the initial set of issues that we sought to resolve in NAVIGATOR.

As we developed NAVIGATOR, we again performed heurisitic evaluations to refine the visual representation and GUI design. Evaluators of the system were all also familiar with GARNET, which helped to ensure that core functionality was preserved and issues with GARNET were being addressed. These evaluations helped to inform many of the design decisions described in previous sections. For instance, the need to keep host groups "in place" as discussed in Section 4.1 came directly from these user evaluations.

8. RELATED WORK

To the best of our knowledge, no other commercial or research tools that use attack graphs model the impact of client-side, credential-based, and trust-based attacks. These are increasingly common attack vectors and without them the threat models of these other tools are now out of date.

Two commercial companies provide attack graph displays. The first, RedSeal [2], has two products (Network Advisor and Vulnerability Advisor) that each match some of NAVIGATOR's functionality. The Network Advisor creates a network map and shows reachability. However, in the network map, all subnets are the same size. This means the map does not give the user a visual idea of how many hosts are in the subnet or how important the hosts are. In the Vulnerability Advisor, the user can see subnets that can be reached from the internet or extranet. Since these are two separate tools, it may not be possible to look at reachability and attacks simultaneously.

The second, SkyBox [3], also has two products (Network Compliance Auditor and Risk Exposure Analyzer) that each match some of NAVIGATOR's functionality. The Network Compliance Auditor creates a network map and simulates network traffic flow. However, similar to RedSeal, the subnets are all the same size and do not immediately give the user information about the hosts on each subnet. The Risk Exposure Analyzer prioritizes vulnerabilities and patches. However, it doesn't overlay these attacks onto the network map, which makes it more difficult for the user to get the big picture.

The work of O'Hare et al. [13] is probably the closest in spirit to our approach. Their system also overlays attack graph data on a collection of subnets. However, the system does not calculate or model reachability, nor does it visualize infrastructure devices. Homer et al. [8] do visualize infrastructure, but their aggregation work is focused on trimming superfluous attack steps. They do not group hosts on the network or otherwise visually aggregate at the network device level.

Visualization of network traffic, e.g. [6, 15], can offer some lessons for the visualization of reachability. However, these systems show what *is happening* or *has happened*, not the comparatively large set of what *could happen*. Tools to visualize firewall rulesets are more in line with our efforts. However, much of that work visualizes the rules or rulesets in isolation [12] and doesn't give intuition into the rules' interactions with the network. It additionally takes on the tack of showing reachability in the context of address space, not network devices – an approach that will become more challenging in the context of IPv6 [4].

9. FUTURE WORK

Now that NAVIGATOR displays infrastructure devices, we would like to allow the user to see the rules that comprise each of these devices. This is one of the last pieces of data that NetSPA requires to build attack graphs that is not shown in NAVIGATOR. Additionally, NetSPA supports

personal firewalls, also known as endpoint or host-based firewalls, that are installed on individual hosts to control incoming and outgoing traffic. Adding a visualization for this in NAVIGATOR would allow users to determine whether their network's personal firewalls have been properly configured.

A potential extension to our hybrid edges could be to utilize a hierarchical edge bundling technique [7] or a flow map technique [14], either of which would result in better merging of edges where width would indicate the number of inbound/ outbound connections. This would allow for a smoother transition between the edge that travels between the subnet and any source side or target side edges.

Our modified strip treemap algorithm works very well at the host-zoom level where it is important that order be maintained. However, it is not always important to maintain the order at the host group level. We would like to explore algorithms that could rearrange the host groups in ways that would further reduce the amount of wasted space.

NetSPA takes as input a snapshot of the network's vulnerability scans and infrastructure configurations. However, a network's security is constantly evolving. NAVIGATOR should be extended to allow users to compare subsequent scans and look at the "delta" of what an attacker can reach. This would allow the user to study the security over time and identify threats that are becoming more prevalent.

Finally, further work can extend our attacker model to include more types of adversaries. We are interested in developing classes of exploits that closely model those available to real world adversaries. For instance, a Metasploit adversary would be able to use any exploit that is available on Metasploit, which includes some server-side and some client-side attacks but not all of either. The user would be able to specify the adversary that they would like to model before creating an attack graph in NAVIGATOR.

10. CONCLUSION

We have developed NAVIGATOR as a tool for visualizing attack graphs and network reachability. It is the first attack graph visualization tool to model and display the effect of client-side, trust-based, and credential-based attacks. NAVIGATOR also greatly enhances our previous GUI, GARNET, by scaling the size of host groups based on asset value, allowing reachability to be seen at the same time as the attack graph, and improving the system's overall speed. Doing this required developing a novel treemap algorithm and a new backend attack graph type. Finally, it displays infrastructure devices and host-level data, so the user can gain a greater understanding of the network.

11. REFERENCES

[1] National vulnerability database, NIST. http://nvd.nist.gov/, March 2010.

[2] RedSeal systems. http://www.redseal.net, March 2010.

[3] Skybox security. http://www.skyboxsecurity.com, March 2010.

[4] D. Barrera and P. van Oorschot. Security visualization tools and IPv6 addresses. In *6th International Workshop on Visualization for Cyber Security (VizSec)*, 2009.

[5] B. B. Bederson, B. Shneiderman, and M. Wattenberg. Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Trans. Graph.*, 21(4):833–854, 2002.

[6] J. Glanfield, S. Brooks, T. Taylor, D. Paterson, C. Smith, C. Gates, and J. McHugh. Over flow: An overview visualization for network analysis. In *6th International Workshop on Visualization for Cyber Security (VizSec)*, pages 11 –19, 2009.

[7] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12:741–748, 2006.

[8] J. Homer, A. Varikuti, X. Ou, and M. A. McQueen. Improving attack graph visualization through data reduction and attack grouping. In *5th International Workshop on Visualization for Cyber Security (VizSec)*, pages 68–79, 2008.

[9] K. Ingols, M. Chu, R. Lippmann, S. Webster, and S. Boyer. Modeling modern network attacks and countermeasures using attack graphs. In *ACSAC*. IEEE Computer Society, 2009.

[10] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *ACSAC*, pages 121–130. IEEE Computer Society, 2006.

[11] R. Lippmann et al. Validating and restoring defense in depth using attack graphs. In *IEEE Military Communications Conference (MILCOM)*, 2006.

[12] S. Morrissey and G. Grinstein. Visualizing firewall configurations using created voids. In *6th International Workshop on Visualization for Cyber Security (VizSec)*, pages 75 –79, 2009.

[13] S. O'Hare, S. Noel, and K. Prole. A graph-theoretic visualization approach to network risk analysis. In *5th International Workshop on Visualization for Cyber Security (VizSec)*, pages 60–67, 2008.

[14] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proceedings of the IEEE Symposium on Information Visualization 2005*, pages 219–224, 2005.

[15] T. Taylor, D. Paterson, J. Glanfield, C. Gates, S. Brooks, and J. McHugh. Flovis: Flow visualization system. *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, pages 186 –198, 2009.

[16] L. Williams, R. Lippmann, and K. Ingols. GARNET: A graphical attack graph and reachability network evaluation tool. In *5th International Workshop on Visualization for Cyber Security (VizSec)*, pages 44–59, 2008.