NV: Nessus Vulnerability Visualization for the Web

Lane Harrison
Oak Ridge National Laboratory
Oak Ridge TN, USA
harrisonlt@ornl.gov

Riley Spahn
Oak Ridge National Laboratory
Oak Ridge TN, USA
spahnrb1@ornl.gov

Mike Iannacone
Oak Ridge National Laboratory
Oak Ridge TN, USA
iannaconemd@ornl.gov

Evan Downing
Oak Ridge National Laboratory
Oak Ridge TN, USA
epdowning@gmail.com

John R. Goodall
Oak Ridge National Laboratory
Oak Ridge TN, USA
jgoodall@ornl.gov

ABSTRACT

Network vulnerability is a critical component of network security. Yet vulnerability analysis has received relatively little attention from the security visualization community. This paper describes nv, a web-based Nessus vulnerability visualization. Nv utilizes treemaps and linked histograms to allow security analysts and systems administrators to discover, analyze, and manage vulnerabilities on their networks. In addition to visualizing single Nessus scans, nv supports the analysis of sequential scans by showing which vulnerabilities have been fixed, remain open, or are newly discovered. Nv operates completely in-browser, to avoid sending sensitive data to outside servers. We discuss the design of nv, as well as provide case studies demonstrating vulnerability analysis workflows which include a multiple-node testbed and data from the 2011 VAST Challenge.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical user interfaces (GUI)*; D.4.6 [**Security and Protection**]: Information flow controls

General Terms

Security, Human Factors

Keywords

security visualization, vulnerability visualization, security analysis, information visualization, cyber security

VizSec '12, October 15 2012, Seattle, WA, USA

1. INTRODUCTION

In order to assess the security posture of the servers and workstations on their network, security analysts and systems administrators use vulnerability assessment tools such as Nessus[1]. Such tools probe machines to determine which network ports are open, what services are running on the ports, and, most importantly, what versions of those services are running. Identifying the services and the versions enables these tools to match them with known vulnerabilities. Nessus and similar tools can produce an overwhelming amount of data for large networks. Traditional reporting tools present the data tabularly, often with color coding to attempt to provide an overview of each vulnerability's severity. But this data can be very large, with little support for comparing individual or logical groupings of machines. Further, it can be difficult to determine whether the overall vulnerability state of a network has changed between scans at different points in time.

Nessus Vulnerability Visualization (nv) was developed to support the discovery, analysis, and management of vulnerabilities in both individual and sequential Nessus scans. The visualizations in nv consist of treemaps and histograms, which facilitate the exploration of vulnerabilities in large networks (see figure 1). Several interactions are provided to help uncover pertinent information to the administrator as they manage large networks.

Nv also allows analysts to specify logical groupings that reflect analyst's situated knowledge [4]. Additionally, nv enables analysts to capture and manage criticality scores for individual and groups of machines in their network. This information is then used to affect size and other visual features in the treemap, which helps ensure that high-value machines receive the most attention.

Specifically, our contributions to the field of security visualization are as follows:

- A visualization tool that supports security vulnerability awareness, analysis, and tracking; and
- A framework for building web-based visualizations that avoid sending sensitive data to servers.

In the following section, we discuss related work in vulnerability analysis and visualization. Afterwards, we discuss the

[1]http://www.tenable.com/products/nessus

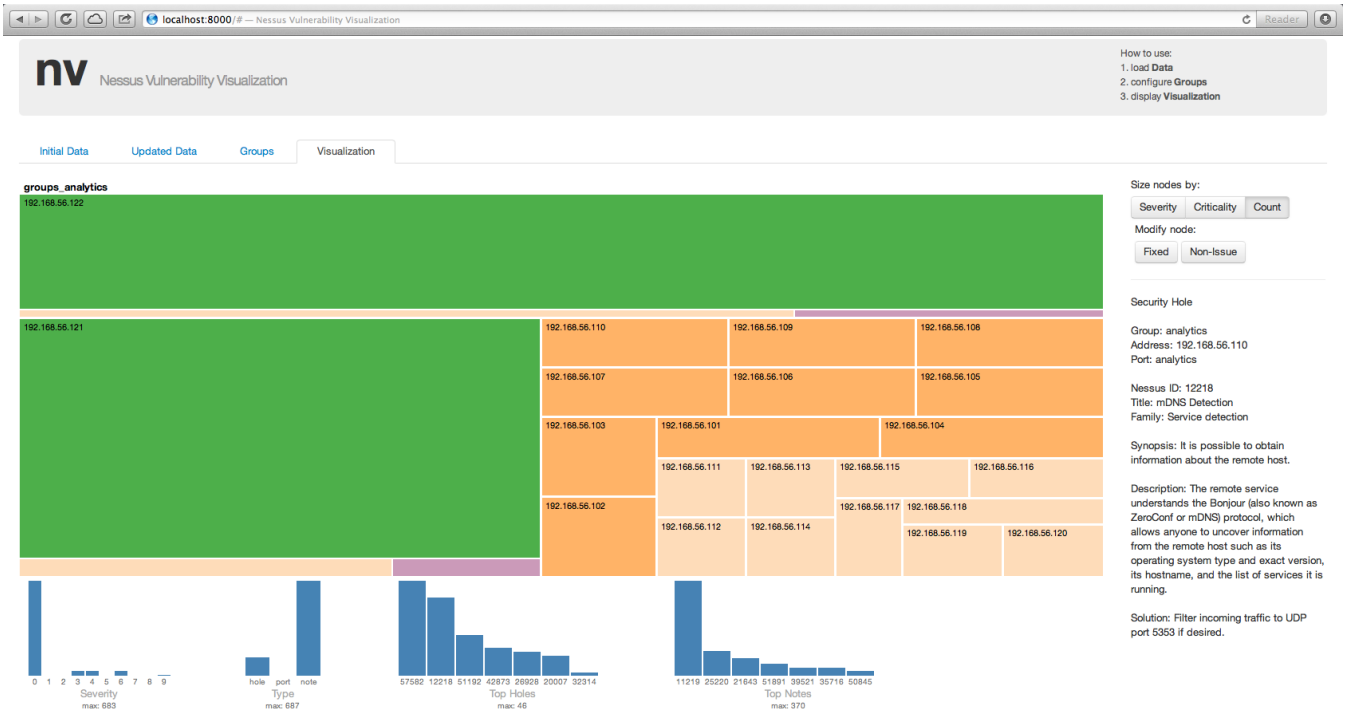

Figure 1: One of the main use cases of nv is to compare two Nessus scans to analyze fixed (green), open (orange), and new (purple) vulnerabilities. Nv runs on the client side and does not require Nessus data to be sent to an outside server.

design of nv and then present several case studies involving Nessus scans from two different test networks. We conclude with a brief discussion on web-based security visualization tools and on our future plans for nv.

2. RELATED WORK

While there has been little research examining ways to depict and explore vulnerabilities, there has been much research in computer vulnerability analysis using graph-based techniques to model the state of the system. Such research addresses a different problem than we are attempting to solve with nv; we are not trying to address identifying all possible paths through a network, but providing an overview of the security state of a network and a means to compare how state changes over time. However, this prior work has somewhat influenced our approach in understanding use cases related to vulnerability analysis.

One such technique is known as Topological Vulnerability Analysis (TVA) [6]. TVA uses the network state and attack vectors between machines to create an attack graph that will model all possible attack paths in a network. To generate these attack graphs TVA uses information from vulnerability assessment tools such as Nessus. These graphs generated by TVA tend to be large, so it introduces an aggregation and visual analysis element to make the models easier to comprehend by an analyst [8]. One aggregation method used by the TVA visualization is to aggregate machines based on their ability to access other machines. A group of machines will be aggregated if each node in the group has access to every other node in that group. These groupings are then aggregated into a single node in the visualization. We also believe that grouping nodes into aggregates is critical to scalability, but our approach relies on analysts' situated knowledge of their networks to create groupings that mirror their own mental models.

Similar to TVA, NAVIGATOR also uses vulnerability assessment data to create and display attack graphs [3]. Like TVA, a central challenge in displaying attack graphs to users is how to aggregate the data. NAVIGATOR utilizes strip treemaps to aggregate machines within each subnet, connected by links displaying potential attack paths through a network.

Researchers have also used model checking tools to manipulate graph representations of a network where each node is a state of the network and each transition represents an exploit [1]. This type of attack graph allows an analyst to focus efforts on patching exploits (edges) that create the largest disconnects in the graph. This type of analysis is convenient because we already have graph algorithms that can efficiently perform such analysis.

Ou, Govindavajhala and Appel take a different approach to security analysis in their MulVAL project [7]. They attempt to model the interactions between known vulnerabilities and software bugs, configurations and permission policies. In their approach an analyst will specify the system and policies in a logic language that is a subset of the Prolog logic programming language and vulnerabilities in the Open Vulnerability Assessment Language. After the systems, policies, and vulnerabilities are defined the MulVAL system uses a two-phase algorithm to simulate both attacks and policy checking. The system generates all possible attacks based on the vulnerabilities and then compares those with the defined policies to detect violations.

These tools attempt to model and visualize all potential

attack paths through a network. While these systems are powerful and integrate much data, they do not attempt to solve the common, straightforward problem of understanding vulnerabilities within large networks. Given enough data, they can solve the problem of tracing potential paths through a network based on those vulnerabilities, but the first step of visualizing vulnerabilities within a network remains an open problem. Additionally, one limitation mentioned by Chu et al. is the need for a tool that compares vulnerability assessments (e.g. what has changed, what has been fixed). We address this directly in nv by allowing users to specify two scans, which are then processed to detect various changes.

Although there has been no visualization research related to visualizing network vulnerabilities, there has been some prior work in related areas. Rather than looking at vulnerabilities that are present within an operational network, some prior work has tackled the problem of code vulnerability analysis [5]. Code vulnerability analysis has several similarities with network vulnerability analysis, the most notable of which is a severity score which can be propagated through functions, classes, and other programming constructs. Similarly, attributes of Nessus scans can be aggregated from vulnerability identification number to port, IP, and any number of higher groups. As such, nv is similar to this work in that it uses hierarchical techniques to visualize data, as well as multiple coordinated views to navigate the data space. The data and use case is very different, however. Rather than focus on software developers, nv targets security analysts; rather than target the development process, nv supports network operations.

Rasmussen et al. explicitly visualize the criticality of systems in a network in [9]. Nv is similar in that it allows the administrator to define groups in the hierarchy and assign either groups or machines a criticality score. This score is then used in the visualization.

While these approaches provide a detailed assessment of the accessibility of vulnerable targets, our goal was to create a more widely applicable system. Nessus is the de facto standard for vulnerability assessment. Rather than build an entire system, we wanted to leverage data that is already commonly used in the security community. This approach increases the potential adoption and value to the security community. In addition to leveraging data that analysts already use to encourage adoption and use, we wanted our system to be usable without installing any software, so our approach is web-based, but does all processing of potentially sensitive data on the client within the browser.

3. SYSTEM DESIGN

The goal of nv is to support security analysts and systems administrators in understanding the security posture of their networks by displaying vulnerabilities on their systems. Nv inputs the results of a vulnerability assessment tool (we use Nessus scans) and (optionally) user defined groupings and criticality of machines in their network to create an interactive visualization. This visualization is designed to support common workflows in vulnerability discovery, analysis, and mitigation. These are described in the Case Studies sections. This section covers the visualization and interaction design.

3.1 Data

The results of a vulnerability assessment tool provide significant detail about the state of all machines on the specified network. This information includes the port the vulnerable service is running on, what service and what version is running, what other versions of this software share this vulnerability, and a general description of the vulnerability. These results also indicate whether this is an actual vulnerability or just a general security notice, and it also provides a severity score and several unique identifiers related to this vulnerability, which can be used to find additional information. The results also often give information about how this vulnerability can be patched or otherwise mitigated. The following shows an example; this particular example is from the VAST Challenge 2011 data set.

```
results|192.168.2|192.168.2.175|cifs
(445/tcp)|46844|Security Hole|

Synopsis :
The remote Windows host contains a
font driver that is affected by
a privilege escalation vulnerability.

Description :
The remote Windows host contains a
version of the OpenType Compact
Font Format (CFF)
Font Driver that fails to properly validate
certain
data passed from user mode to kernel mode.
By viewing content rendered in a specially crafted
CFF font, a local
attacker may be able to exploit
this vulnerability to execute
arbitrary code in
kernel mode and take complete control of the
affected system.

Solution :
Microsoft has released a set of patches
for Windows 2000, XP, 2003,
Vista, 2008, 7, and 2008 R2 :
http://www.microsoft.com/
technet/security/Bulletin/MS10-037.mspx

Risk factor :
High / CVSS Base Score : 9.3

(CVSS2#AV:N/AC:M/Au:N/C:C/I:C/A:C)

Plugin output :
- C:\\WINDOWS\\System32\\Atmfd.dll has not been patched
Remote version : 5.1.2.226
Should be : 5.1.2.228

CVE : CVE-2010-0819
BID : 40572
Other references : OSVDB:65217,MSFT:MS10-037
```

3.2 Use Case

The primary goal of nv is to support an understanding of the vulnerabilities within a network to assist decision-making in determining how to allocate and prioritize limited resources to reduce the security vulnerability of high-value targets within their network.

Specifically, the main tasks that nv seeks to support are:

- Identifying the individual machines that have the most severe vulnerabilities;
- Discovering the services that have the most vulnerabilities within their network;
- Identifying the exposure to vulnerabilities within groups of machines where those groupings reflect the analyst's mental model of their network;
- Determining the high-value machines that are vulnerable to exploitation; and
- Comparing point-in-time snapshots of the security state of the machines in the network, and understanding the differences between these two points.

3.3 Visualization and Interaction

Nv consists of multiple coordinated views including a treemap, several histograms, and a detail-information area showing information on the selected Nessus id. Each of these are designed to support a specific aspect of the vulnerability analysis workflow.

Our primary visualization is a zoomable treemap[10]. We chose to use a treemap over other hierarchical visualization methods such as network/tree-layouts for several reasons. First, our goal with nv is to support the analysis of vulnerability assessment scans on large networks. While information on the network topology is useful for vulnerability analysis, it is important to note that in large dynamic networks, a complete network topology is often either unavailable or too large to be visualized directly. The space-filling aspects of treemaps make them more scalable in this regard.

The treemap allows us to easily identify machines with the most severe vulnerabilities. The administrator's eye will be naturally drawn to the largest and darkest-colored nodes in the treemap, revealing the machines with the largest amount of severe vulnerabilities. The treemap allows the user to group machines together in a way that makes sense to them personally. Oftentimes the analyst has a mental picture of how their network is laid out and benefit from using a flexible environment that lets them customize it as they see fit. Again, using custom grouping, the user can easily group high-value machines into a single group and can be monitored closely for vulnerabilities, or organize their groupings according to mission or common roles.

Allowing users to compare two different points in time of their network is also crucial. They can use these snapshots to not only monitor how efficiently the network is being patched, but also see how new vulnerabilities have been introduced over a period of time. If there are more new nodes than fixed nodes, then they can assume that either attention to that group of machines has been depleted or that the group has become more susceptible to vulnerabilities over time. All of these analyses are important to security analysts and systems administrators and can help them to monitor and protect their network more efficiently and effectively.

Another reason we used treemaps was for their ability to effectively make use of both size and color for encoding data attributes. Since Nessus data is not stored in a hierarchical form, it could be visualized using many multi-dimensional

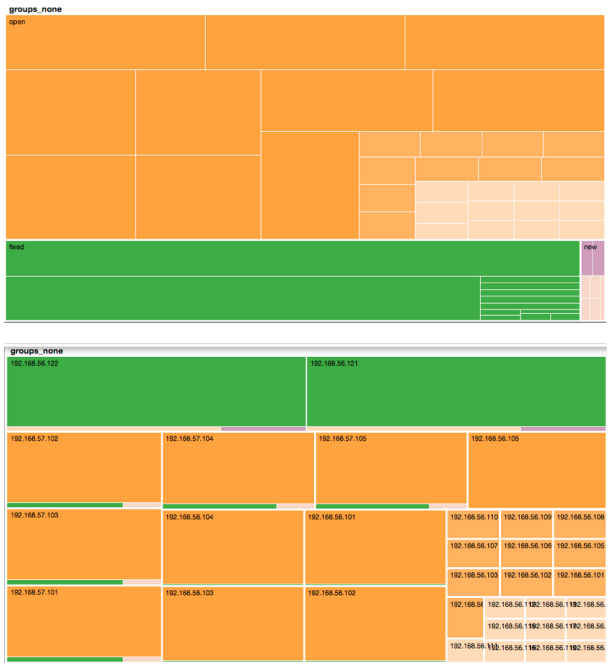

Figure 2: Nesting on different values gives different results. Top: splitting by vulnerability-state before splitting by IP. Bottom: the same data, but split by IP before vulnerability-state.

visualization techniques, such as parallel coordinates or scatterplot matrices. However, because the scalability of the visualization and the ability to mirror users' mental models were primary concerns, we opted to nest the data from individual vulnerabilities and ports up to IPs and groups of IPs.

We also use data-accumulation and coloring methods to ensure that data is not obscured by the hierarchy. For instance, when comparing two vulnerability scans, nodes are colored by the maximum count of vulnerability states (fixed, open, or new vulnerabilities) in their child nodes. A potential disadvantage of this approach is that a node could contain slightly more fixed vulnerabilities than open vulnerabilities, and yet will still be colored green. To alleviate this problem, we add the option to split the nodes by vulnerability-state higher in the hierarchy; figure 2 shows splitting nodes by vulnerability-state and by IP.

The advantage of separating vulnerability-states higher is that the analyst can explore only the fixed vulnerabilities or only the open vulnerabilities. However, the disadvantage of this approach is that the IPs are then separated since they can appear in any branch of the hierarchy (fixed, open, and new). To our knowledge, there exists no widely accepted visual technique that can effectively represent multiple attributes at every level in a treemap. However, we plan to explore other common approaches such as glyphs and combined color scales in future work.

Since analysts can specify the criticality of both individual machines and groups of machines in nv, the treemap includes sizing by criticality as an option. The most critical machines therefore appear as larger nodes, while still being colored by severity. Other sizing options include severity (the default)

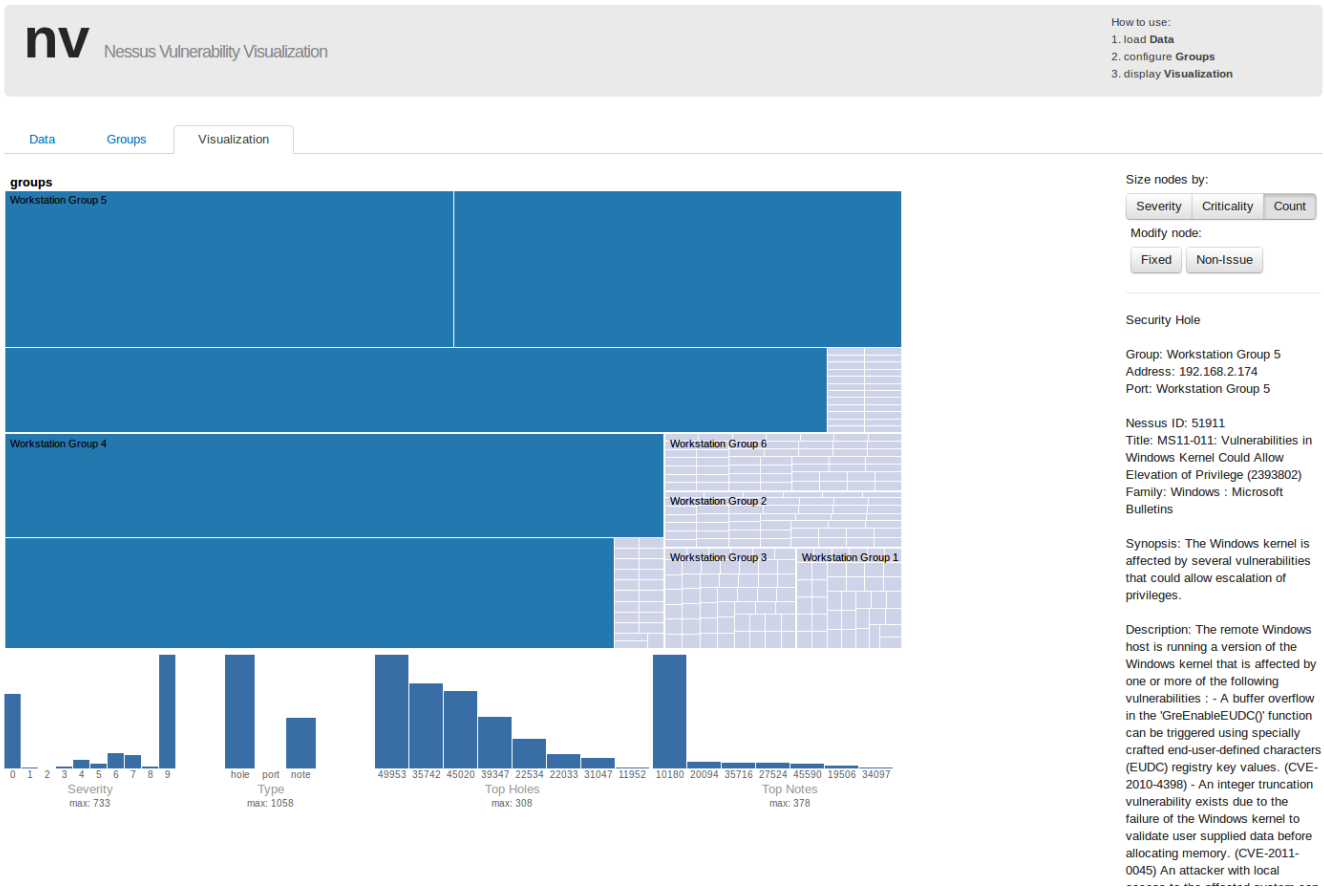

Figure 3: Top level view of the large VAST Challenge 2011 dataset. The prevalence of dark blue indicates numerous vulnerabilities with high severity scores on Workstation Groups 4 and 5.

and by vulnerability counts. Dual encoding severity with both color and size can be useful, as the darkest colored and largest nodes appear at the top left in each level of the histogram.

The color scales in the treemap were created using ColorBrewer2[2]. While the primary color scales shown in the paper are designed to have semantic meanings (green for fixed, red for new, orange for open), we also follow visualization best practices by including a colorblind-safe version.

Nv includes several histograms, including vulnerability-type (note, hole, or open port), severity (CVSS score), top Nessus note ids, and top Nessus hole ids. These histograms serve dual purposes, as both overviews of the data and as filters by which users may guide their analysis. For instance, by brushing over the highest values in the severity histogram, the appropriate nodes in the treemap are highlighted. This works by examining each child of each element in the current level of the hierarchy. Another use of the histograms is to easily highlight the most commonly occurring vulnerabilities in the network. A possible drawback of this approach is that sometimes the least common vulnerabilities can be the most damaging. However, this issue is mitigated by the fact that the treemap can be be sized and colored by severity, which makes the most damaging vulnerabilities easy to find. The histograms operate as conjunctions (AND), meaning that the user can specify queries such as all vulnerabilities of type hole with severity of 5 or greater.

The Nessus information area is updated when the user drills down to the level at which Nessus vulnerability identification numbers are shown. The area then updates with detailed information about the currently selected Nessus id, including a synopsis, detailed description, vulnerability family, and solution (when available). Based on this information, the user has the option to mark the vulnerability as either fixed or as a non-issue, which re-colors the node in the treemap. This functionality is intended to serve as a way for analysts to avoid revisiting vulnerabilities that have been addressed in the scan being examined.

3.4 Implementation

One significant requirement for this project was to not unnecessarily disclose the vulnerability assessment scan results to any third parties; because this information would be very valuable to any attacker, the users of this tool would have an obvious concern to prevent its disclosure. To address this concern, nv runs entirely in the browser client, without relying on any server-side functionality, and without loading any non-local resources. We also developed a custom parser for the data files, and related code to compare and merge these results. We were also able to handle these tasks in the browser and maintain interactive performance. Additionally, the entire technology stack is open source, and the

[2]http://colorbrewer2.org/

tool itself will also be open sourced.

One difficulty caused by the requirement of not leaking scan results was how to look up additional details about the results. Nessus provides an interface to access significant additional information about any specific vulnerability ID, including useful details such as related CVE and Bugtraq IDs, and information about how to patch or otherwise address each vulnerability. However, using this directly could still give an adversary significant information; if they could observe any of this traffic, then they could still learn which vulnerabilities are present. To address this, we build a local cache of this information, which the client can access offline.

The main treemap and the histograms were created using the d3 library [2], which is designed for "apply[ing] data-driven transformations" to the Document Object Model (DOM). D3 is fast, flexible, and supports large datasets, which were our main requirements. Crossfilter[3], designed for accessing "large multivariate datasets in the browser", was used to store and access our Nessus scan results and all related information about the machines and subnets on the network. This handles the data entirely in memory, and handles storage and access in an efficient manner.

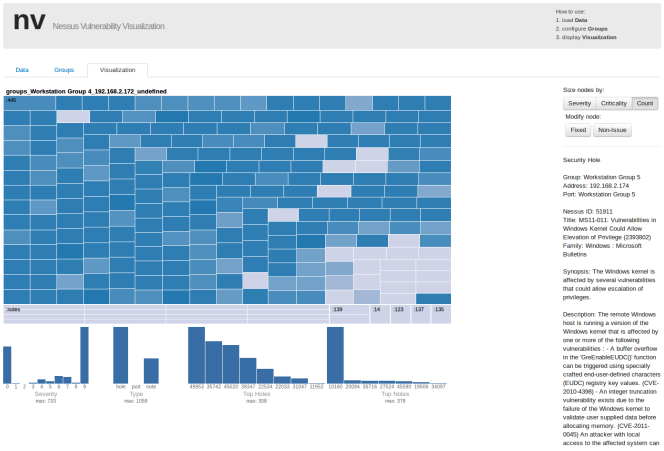

Figure 4: Port level view of an All Freight Corporation workstation. Port 445 is shown to have numerous severe vulnerabilities. Information for selected vulnerablities is shown on the right.

4. CASE STUDIES

We envision nv as being useful for two types of use cases: The first is to analyze the current vulnerability state of all machines on a network. This use case is to allow a user to prioritize maintenance based on the value of the machines and the criticality of the vulnerabilities found on those machines using data from Nessus scans. The second use case is visualizing the changes to the vulnerability states of machines on a network after a systems administrator performs maintenance to determine if the maintenance was effective at addressing system vulnerabilities.

4.1 Case Study: Static Vulnerability State Network

To test visualizing a static network, we used Nessus scan data from the VAST Challenge 2011[4]. This data is from a simulated network for the fictitious All Freight Corporation. The VAST Challenge gives us an established network dataset to test how nv scales to a large data set that contains many vulnerabilities spread across a variety of machines and groups. This data set has more than one hundred-fifty unique IP addresses associated with various workstations in the scan. The Nessus scan shows that numerous machines on the network have some sort of security hole, such as incorrectly configured telnet client, a font driver that allows privilege escalation, and a vulnerability in an outdated version of Microsoft Excel.

We split the workstations into six groups with criticality scores ranging from two to nine. The major security holes in the group are concentrated in group four with a criticality score of nine and in group five with a criticality score of two. When the security analyst looks at the groups level on the tree map it is immediately obvious where their attention is needed most. As shown in figure 3, the analyst can see that there are many high-scoring vulnerabilities that will require attention right away. Workstation groups four and five dominate the treemap in all three visualization modes and take up more than half of figure 3. The analyst can also use the histograms at the bottom of the visualization to quickly get an overview of the network's current vulnerabilities. The histograms show that there a large number of high severity vulnerabilities that need urgent attention. When the analyst zooms into group four, they see that most of the vulnerabilities are located on two IP addresses and when they select IP address 192.168.2.172, they see that nearly all of the vulnerabilities are associated with port 445 and a Windows file sharing program. corollary 4 shows the port level view of 192.168.2.172. Nv uses color gradients to make the most critical and severe vulnerabilities most prominent in the visualization. In this view the analyst sees that port 445 has numerous potential vulnerabilities that range from light blue and insignificant to dark blue and critical. After this analysis an analyst could conclude that he should focus his efforts on the Microsoft file sharing programs that utilize port 445.

The analyst can also explore the other visually dominant IP address, 192.168.2.171, and see that this machine's vulnerabilities come from port 139 and NetBIOS. This exploration allows the analyst to easily discover vulnerabilities in the system and prioritize repair accordingly. It also makes it easier to view large networks because the IP addresses are aggregated into nodes that can be expanded to view the individual IP addresses contained in that group. In the VAST Challenge, these two machines become the source of much of the malicious activity on the network. Accordingly, a tool such as nv could be used to make analysts aware the state of these machines before the malicious activity begins.

4.2 Case Study: Dynamic Vulnerability State Network

Another use case for nv is to make it easier for users to visualize and compare the state or a network at two points in time such as before and after performing maintenance. In this example, machines are grouped into three different categories: One group is a set of workstations split between two hundred fifty-six Fedora workstations and three hundred twenty Ubuntu workstations. The second group is a set

[3]http://square.github.com/crossfilter/
[4]http://hcil.cs.umd.edu/localphp/hcil/vast11/

Name:Criticality	Machine Count	IP Addresses	Time Period	Security Notes	Security Holes
Workstations:2	576	192.168.5[6|7|8].x	Before Maintenance	8320	1024
			After Maintenance	2944	0
LAPP Servers:9	256	192.168.59.x	Before Maintenance	10496	256
			After Maintenance	10240	0
Wordpress Servers:5	256	192.168.60.x	Before Maintenance	9984	256
			After Maintenance	9984	0

Table 1: The number of security notes and security holes by grouping before and after maintenance.

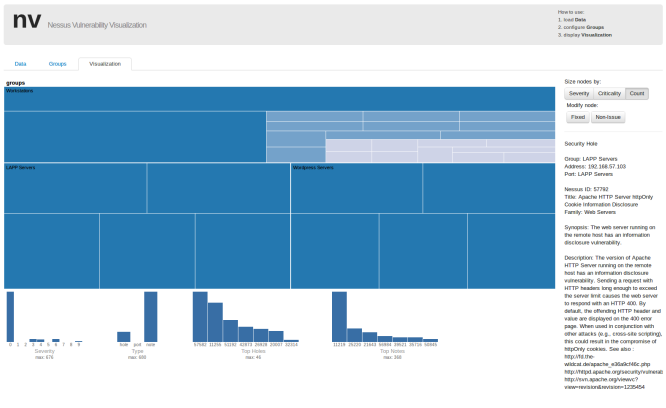

Figure 5: Top level view showing the user-defined IP address groups.

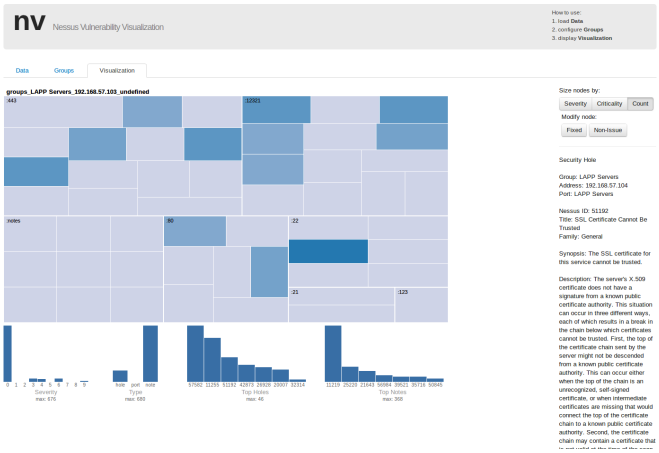

Figure 6: After sizing by severity, the view shows vulnerabilities distributed across the LAPP Server's ports.

of two hundred fifty-six machines that serve the Wordpress blogging software. The last grouping is a set of two hundred fifty-six Linux Apache PostgreSQL PHP (LAPP) servers. Initially all of these groupings contain serious vulnerabilities. The LAPP servers are running a poorly configured file transfer protocol (FTP) server and both the LAPP and Wordpress servers have simple root passwords that Nessus shows as a security hole. The majority of the workstations are properly configured except for sixty-four that contain multiple security holes. These workstations are running outdated versions of the Ubuntu operating system and have vulnerabilities such as an FTP server that allows a remote user to execute arbitrary code, an incorrectly configured Windows file sharing software, weak secure shell (SSH) keys and a Samba server that is vulnerable to buffer overflow attacks.

While in the top level visualization mode the user's attention is drawn to the large LAPP server node. The size is an indication of the importance of the situation based on the number of security holes discovered, the severity of the security holes discovered, and the assigned criticality of the machines in the group, see figure 5. When zooming into the LAPP Server branch of the treemap it becomes clear that all five of the machines seem to be equally at risk. To gain further insight, the user zooms into the node for a specific machine where each node represents a port with an associated vulnerability in figure 6. At this specific port node the administrator can click on a vulnerability ID and the tool will display information about the vulnerability and potential solutions in the right-most panel of the tool. In this situation the LAPP servers all have the same untrusted SSL certificate security hole.

When the user zooms back out to the group view and switches the visualization to severity mode, as in figure 6, the workstation's node grows bringing it into greater prominence. When zooming into the workstation group, they can see that a subset of the IP addresses have much larger and darker nodes than any of the other workstations. If they zoom into one of these IP addresses, they see that many of the most severe vulnerabilities are associated with ports 445 and 80. The user can examine each port node's child, seeing information about the specific vulnerabilities in the right-most panel, discovering that the machine is running a poorly configured Apache Web Server and that a Windows share that can be accessed through the network.

After further exploring the network, the administrator patches the most critical vulnerabilities in the system. Table 1 shows the number of security notes and holes before and after maintenance. Nv provides functionality to compare two vulnerability assessment scans to show changes between two states, such as before and after applying patches. After patching high value systems the administrator can rescan the network, then explore and see the differences between the previous state and the newly patched system.

Nv shows corrected vulnerabilities in green, the remaining vulnerabilities in orange, and any new vulnerabilities in purple, as in figure 7. The systems administrator can easily see that the major workstations vulnerabilities have been patched. Zooming into the workstation node the system administrator sees that while they were patching the most severe vulnerabilities they inadvertently opened new vulnerabilities on the two machines and did not address some of the vulnerabilities seen earlier.

We simulated this use case using virtual machines (VM)

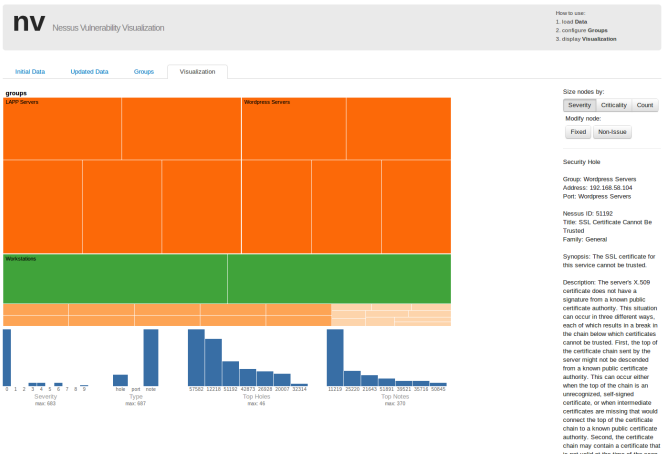

Figure 7: After fixing previous vulnerabilities, subsequent Nessus scans are processed to identify open (orange), new (purple), and fixed (green) vulnerabilities.

communicating through a host-only network. Using a host-only network allowed us to use Nessus from the host to scan the VMs. We used one grouping of two different types of work station and two groupings of similar servers. Both groups of servers were using Ubuntu 10.10 LTS. Ten of the Ubuntu workstations were using Ubuntu 11.10 while the two workstations with the massive number of vulnerabilities were using Ubuntu 8.04 with purposely unpatched and misconfigured software. The Fedora workstations were running Fedora 15. We used the Metasploitable virtual machine image to simulate the two vulnerable workstations before they were upgraded to 11.10.

To reflect real-world scenarios, we did not patch all vulnerabilities in this use case. Instead, the systems administrator would only handle the most important vulnerabilities and system updates. In this simulated use case we improved the weak root passwords and corrected the poorly configured FTP server seen on the servers. We focused on updating and correcting the two most vulnerable workstations by updating them to be at the same vulnerability level as the other ten Ubuntu workstations.

5. CONCLUSION AND FUTURE WORK

We have introduced nv, a Nessus vulnerability visualization system. Nv is designed to support security analysts and systems administrators in the tasks of vulnerability discovery, analysis, and management through an interactive visualization. This tool abstracts and aggregates the large amounts of data produced by vulnerability assessment tools like Nessus so as not to overwhelm the user. Nv is also designed to protect the privacy of users through client side computation, since sending sensitive data, like vulnerability scans, over a network introduces an unnecessary vulnerability and potential attack vector. We have also provided case studies, which suggest that nv is appropriate for a range of common vulnerability analysis tasks. Finally, nv will be made available online and released as open-source.

6. ACKNOWLEDGEMENTS

This work was supported by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract no. DE-AC05-00OR22725.

7. REFERENCES

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proc. ACM Conference on Computer and Communications Security*, pages 217–224, 2002.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. In *IEEE Trans. Visualization and Computer Graphics*, volume 17, pages 2301–2309, 2011.

[3] M. Chu, K. Ingols, R. Lippmann, S. Webster, and S. Boyer. Visualizing attack graphs, reachability, and trust relationships with NAVIGATOR. In *Proc. Symposium on Visualization for Cyber Security (VizSec)*, pages 22–33, 2010.

[4] J. R. Goodall, W. G. Lutters, and A. Komlodi. I Know My Network: Collaboration and Expertise in Intrusion Detection. In *Proc. ACM Conference on Computer-Supported Cooperative Work (CSCW)*, pages 342–345, 2004.

[5] J. R. Goodall, H. Radwan, and L. Halseth. Visual analysis of code security. In *Proc. Symposium on Visualization for Cyber Security (VizSec)*, pages 46–51, 2010.

[6] S. Noel, M. Elder, S. Jajodia, P. Kalapa, S. O'Hare, and K. Prole. Advances in topological vulnerability analysis. In *Proc. Cybersecurity Applications and Technology Conference For Homeland Security (CATCH)*, pages 124–129. IEEE, 2009.

[7] X. Ou, S. Govindavajhala, and A. W. Appel. Mulval: a logic-based network security analyzer. In *Proc. Conference on USENIX Security Symposium*. USENIX Association, 2005.

[8] K. Prole, J. R. Goodall, A. D. D'Amico, and J. K. Kopylec. Wireless cyber assets discovery visualization. In *Proc. Workshop on Visualization for Computer Security (VizSec)*, pages 136–143. Springer-Verlag, 2008.

[9] J. Rasmussen, K. Ehrlich, S. Ross, S. Kirk, D. Gruen, and J. Patterson. Nimble cybersecurity incident management through visualization and defensible recommendations. In *Proc. Workshop on Visualization for Computer Security (VizSec)*, pages 102–113, 2010.

[10] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. on Graphics*, 11(1):92–99, 1992.