AlertWheel: Radial Bipartite Graph Visualization Applied to Intrusion Detection System Alerts

Maxime Dumas, Jean-Marc Robert, and Michael J. McGuffin, École de Technologie Supérieure

Abstract

Intrusion detection systems, or IDSs, are network security tools that generate huge quantities of information which are challenging to analyze. Information visualization is essential for efficiently parsing these data to discover the underlying causes of computer security breaches. AlertWheel is a user interface featuring a novel radial overview visualization, as well as filtering, drilling down, and saving and annotating subsets of data, to support the workflow of real network defense analysts. In designing AlertWheel, we identified new ways of displaying bipartite graphs (i.e., network diagrams showing links between two sets of nodes). The links in AlertWheel's visualizations are positioned and shaped to avoid occlusion of data, and three different edge bundling techniques are used to reduce clutter.

Computers exposed to the Internet are regularly scanned and attacked by outsiders. Intrusion detection systems (IDS) monitor and record such activity by capturing all incoming network events (or *alerts*) in a multidimensional dataset that stores the source, time, and nature of each event. Automated means are available to analyze such data; however, visualization of the data by human users is also valuable for security experts to gain understanding and insight into network activity, as well as to communicate with other stakeholders, and eventually adapt security strategies.

Data from an IDS can be visualized using various previous techniques. At one extreme is the approach of SnortView [1], which essentially enumerates individual events in a table, allowing all details to be seen but scaling poorly to large datasets. At another extreme are techniques that reduce events to individual pixels [2], allowing many events to be seen at once, but sharply constraining how much information the user can see about one event without switching to a more detailed view. Somewhere in the middle of this spectrum is VisAlert [3], which uses a radial visualization [4] to show the location, time, and nature of many events at once.

We present AlertWheel (Fig. 1), which further explores the middleground of this spectrum. Like VisAlert, AlertWheel also uses a radial visualization, and also shows the location, time, and nature of many events at once. However, AlertWheel's radial layout is better suited for visualizing attacks against a honeypot [5], our driving application. Visualizing such attacks is useful for network analysts during the outbreak of a virulent malware on the Internet. Furthermore, our work has led to new ways of drawing bipartite graphs (Fig. 2) that are visually clearer than status quo approaches and can be applied to other application domains. We also discuss how the design of AlertWheel is motivated by the workflow of network defense analysts [6].

Visualizing Bipartite Graphs

Each event or alert generated by an IDS has an associated source (address of the outside machine which initiated the event) and category. For example, Snort is a common, open source IDS that defines over 30 categories of alerts ("network-scan," "denial-of-service," "trojan-activity," etc.). One way to visualize such data is as a diagram where each alert is shown as a link (or edge) connecting a source address to a category. This kind of diagram corresponds to a *bipartite graph*, that is, a graph made of two sets of nodes (source addresses and categories) where all edges connect a node in one set to a node in the other set.

VisAlert [3] displays a similar bipartite graph, showing categories on the outside connected to locations on the inside of a radial layout. In their case, the locations shown are destination addresses (addresses of machines inside a network being monitored), allowing VisAlert to show network topology in the center of the visualization. However, in AlertWheel, we want to visualize source addresses in the outside world, and these can be quite numerous. We therefore place addresses on the *outside* of our radial layout, where there is more room, and put categories on the *inside*, since they are much fewer in number. Furthermore, VisAlert draws links as straight line edges that can lead to much occlusion. AlertWheel instead draws links as carefully designed curves and uses three different bundling techniques to clarify them.

As seen in Fig. 1, the outer nodes of the radial layout are XS13334, XS40066, and so on, which are anonymized autonomous system (AS) nodes, corresponding to ranges of IP addresses of source locations, or "bundles" (groupings) of such AS nodes. On the inside of the layout are colored pie slices representing categories of alerts: the largest yellow slice is "misc-activity." Colors show the severity of the alert category (red being the most severe). Each link from an AS node to a category corresponds to at least one alert, with the thickness

0890-8044/12/$25.00 © 2012 IEEE

of the link proportional to the logarithm of the number of alerts. As shown in Fig. 3 (left), each link is made of two stems connected by a circular midsection (all three parts aligned with a polar coordinate system). To make the edges easy to distinguish, the radius of the circular midsection is varied according to the node it is connected to. In Fig. 1, there are six categories, hence six different radii for the circular midsections. As can also be seen, each outer stem can split and merge with multiple circular midsections at different radii; each circular midsection can be contributed to by multiple outer stems, each circular midsection leads to a single inner stem, and each inner stem can be contributed to by circular midsections at only one radius.

Edge bundles are groups of edges that merge and/or split, and have been used to reduce clutter in diagrams [8–10]. Examples are shown on the right in Fig. 3. For the purpose of discussing previous work, we distinguish bundles at the stems (or endpoints) of edges, and bundles at the midsections of edges. In some previous work [8], midsection bundles are used in a way that creates ambiguity. For example, in Fig. 3 (right), it is unclear if node A is connected to node C, D, or some other node. Other previous work has used midsection bundles in unambiguous ways: confluent drawings [11] use midsection bundles to replace the edges between bicliques (i.e., complete bipartite subgraphs). If the midsections in Fig. 3 (right) followed the conventions of confluent drawings, it would imply that A is connected to *both* C and D, and B is also connected to both C and D. Another type of bundling found in previous literature is what we call stem bundling [9], which by its nature cannot create ambiguity. The work of [10] is the only previous work we know of that uses both stem bundles and midsection bundles; however, it avoids introducing ambiguity by not completely overlapping edge segments; instead, the user may zoom in on bundles to resolve individual edges.

The radial layout in AlertWheel is unique in that it uses both stem bundling and unambiguous midsection bundling, with edge segments truly overlapping (thus saving space). In addition, AlertWheel also uses a third kind of bundling which we call *neighborhood bundling*. This is seen in Fig. 1, in the outer nodes that are shaded in gray: each gray node represents a group of nodes that have the same neighbors on the inside of the layout. For example, "Bundle (57)" near the top of the layout is a group of 57 different AS nodes, all of which are originators of events of the same two categories: "misc-activity" (the largest pie slice) and "shellcode-detect" (the third largest slice). Drawing a single stem for this group of 57 nodes greatly simplifies the visualization without introducing any ambiguity. Such neighborhood bundling has been called *edge compression* in previous work [12], although it has not previously been combined with other bundling techniques. Figure 4 shows the same data as Fig. 1, but without neighborhood bundling, illustrating the effect it has on clutter. Figures 2g and 2h also show the difference, before and after, neighborhood bundling. In Fig. 2, each of the nodes C, E, and F have the same neighbors {G, H, J}, and hence are grouped together in Fig. 2h.

We note that 300 nodes on the outer circle is close to the maximum of what the radial overview can show, assuming a diameter of 1000 pixels (close to the maximum vertical resolution on typical displays) and assuming a minimum desired size of about 10 pixels per node ($1000 \times \pi/10 \approx 300$). Fortunately, neighborhood bundling helps avoid this limit.

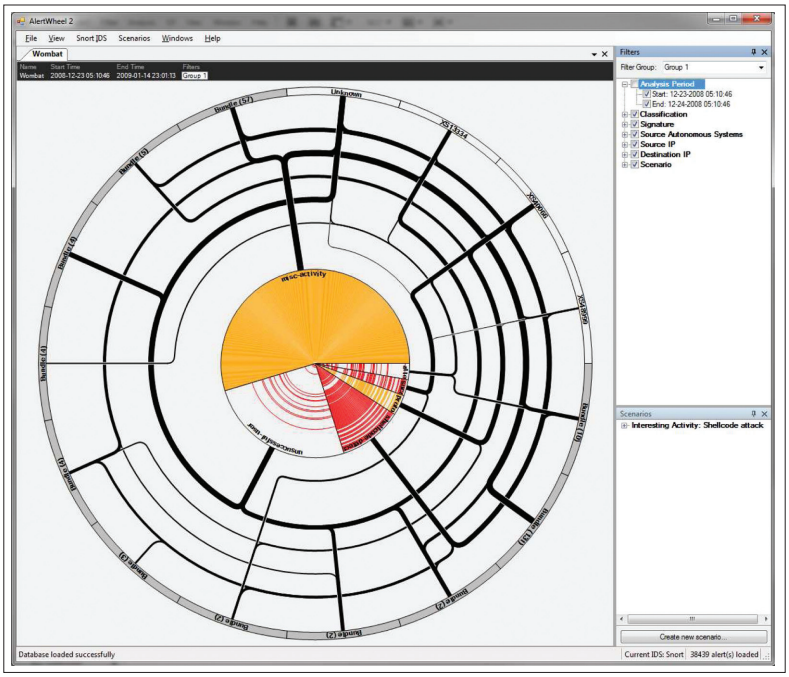

Figure 1 *The AlertWheel user interface. The main view shows the radial visualization of information. At the upper right is a panel for defining filters (used to generate subsets of data), and at the lower right is a panel that lists user-defined "scenarios" (i.e., saved subsets of data).*

In AlertWheel, there is one circular midsection bundle for each inner node (4 in Fig. 2g, 6 in Fig. 1). We could have instead made each midsection bundle correspond to one outer node, but this would imply a greater number of distinct radii for the midsection bundles (and hence more crowding), since the outer nodes are more numerous. Details on the positioning of edges and outer nodes are reported in [13].

Additional Features of AlertWheel

As already mentioned, the pie slices at the center of Figs. 1 and 4 correspond to alert categories, each of which is colored by severity. The angle covered by the pie slice is proportional to the number of alerts for that category, and pie slices are ordered by angle. Each pie slice also contains circular arcs indicating the time of each alert, with the center corresponding to the earliest possible time, and the outer circumference of the pie slices corresponding to the latest possible time. It is possible for the user to click on a pie slice to expand it, revealing the IDS signatures corresponding to the alerts in that pie slice's category.

It is also possible for the user to create multiple subwindows within the main window (Fig. 5), each of which contains a radial overview or a detailed list view. Having multiple radial overviews is useful for comparing different subsets of data, for example, activity on different days. The user may also configure one radial overview to show source locations on the outer circle, and configure another radial overview to show destination locations on the outer circle, as shown in Fig. 5. Each of the subwindows can have a filter applied to it to show only a subset of alerts. Filters are defined in the upper right panel of the main window (visible in Figs. 1, 5, and 6). Each filter is formed of one or more clauses, and multiple filters can be defined. Each filter can also be shared across multiple subwindows (to view the same subset of data in different subwindows), or each subwindow can have a different filter applied to it.

With three subwindows, the upper left subwindow shows

source AS nodes on the outside of the radial layout; the upper right subwindow shows the same data but with the destination IP address shown on the outside of the radial layout. Together, these two views allow the user to simultaneously see the source, category, and destination information for the alerts. The bottom subwindow shows the detailed list view of the same data.

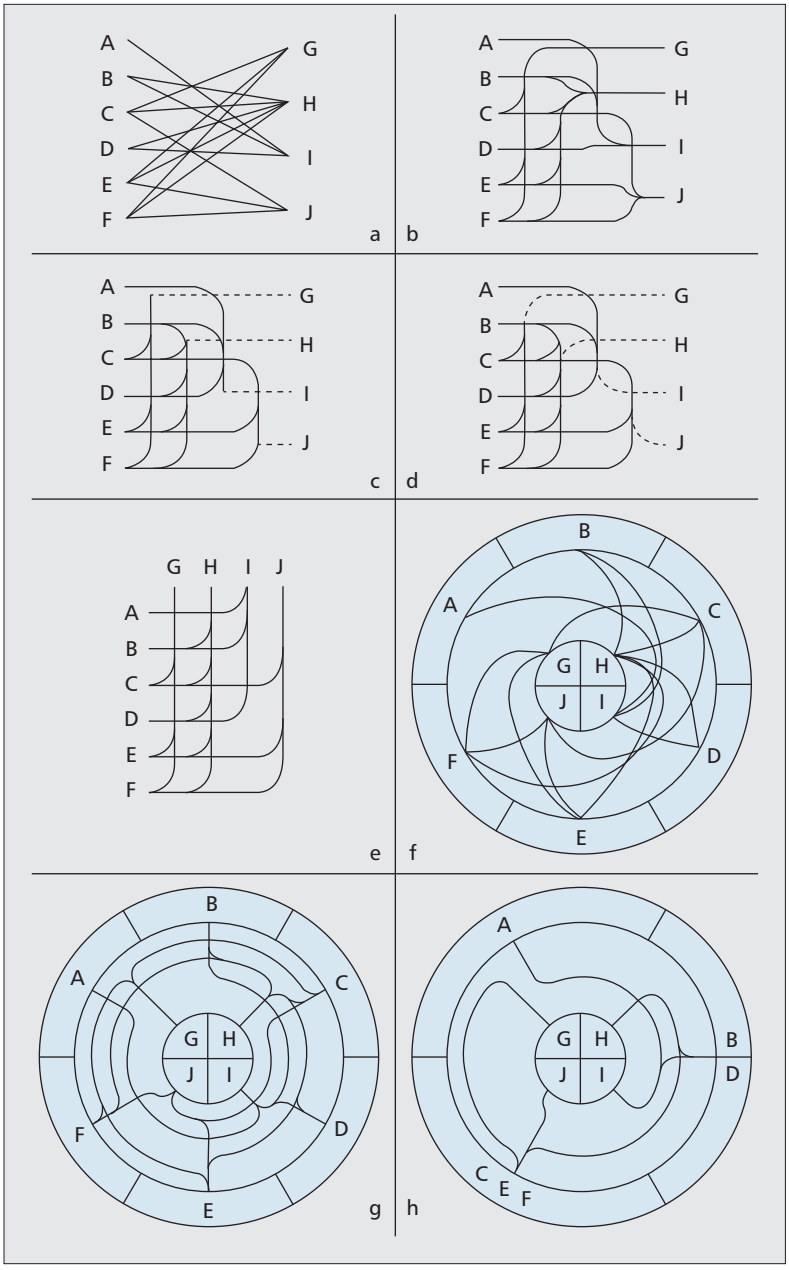

Figure 2. *The same bipartite graph shown in different ways: a) status quo approach; b) each edge starting on the left leads to a vertical bundle, and each vertical bundle leads to a single node on the right; c, d) variations on (b) that make the connections from vertical bundle to node on the right more visually distinct; e) a variation inspired by [7, Fig. 3], where each node on the left corresponds to a horizontal bundle, and each node on the top corresponds to a vertical bundle; f) a radial layout with a naïve way of drawing edges that scales poorly; g) an improved way ofdrawing edges: each edge starting on an outer node leads to a circular bundle, and each circular bundle leads to a single inner node; h) a further improvement that groups together nodes having the same neighbors.*

The clauses available to define a filter are: time interval, "classification" (i.e. the alert categories), signature, source AS node, source IP address, destination IP address, and scenario (which we explain later). For example, a filter could be defined that covers alerts between December 23 and December 24, 2008, *and* are from AS node XS40006, *and* originate from IP addresses which are themselves originators of "misc-activity" *or* "shellcode-detect" category alerts. These filters are defined in a tree menu of checkboxes in the filter panel. To limit the size of the tree menu, rather than listing all possible dates, IP addresses, and so on, the tree menu only displays values explicitly added by the user.

Within each subwindow, the user may select one category (shown in green in Fig. 5), or one AS node (shown in green in Fig. 6). This causes links not connected to that node to be almost completely faded out, allowing the user to visually filter links through selection. Each subwindow has its own selection state. At the same time, rolling the cursor over nodes, without selected them, causes them to highlight with a "datatip" providing more details, and the highlighting is coordinated across all subwindows, to reinforce the conceptual links between them.

The detailed list view (Fig. 5, bottom subwindow) can be placed in a subwindow or can take up the entire window, and can also have a filter applied to it, just as with the radial overview. Each detailed view also has highlighting coordinated with other subwindows. The columns in the detailed view are: source IP address, alert, time, status (which we explain later), severity (low, medium, or high), and destination IP address. The user can sort the list by any column and scroll the list. The resulting enumeration is comparable to SnortView [1], but with some differences. Most significantly, we show the mapping from source IP addresses to alerts, and from destination IP addresses to alerts, as bipartite graphs, and employ another novel way of drawing the connecting edges, shown in Fig. 2c. This reduces clutter and clarifies which alerts share the same source or destination IP addresses (Fig. 5, bottom subwindow). Note that this situation is somewhat simpler than in Fig. 2c, though, because each alert corresponds to only one source (and destination) IP address, meaning that the vertical edge bundles can be ordered in such a way that the source (and destination) IP address can have very short stems without occluding any other vertical bundles.

Design Principles Behind AlertWheel

VisAlert [3] proposed a W^3 principle, whereby the visualization should show *what* event happened *when* and *where*. This is supported by AlertWheel: the event category shown in the center of the radial overview answers the "what" question, the arcs within each pie slice indicate "when," and "where" is indicated by the source location on the outside of the radial overview. All three questions are also answered within our detailed list view. Note also that, with the Aler-

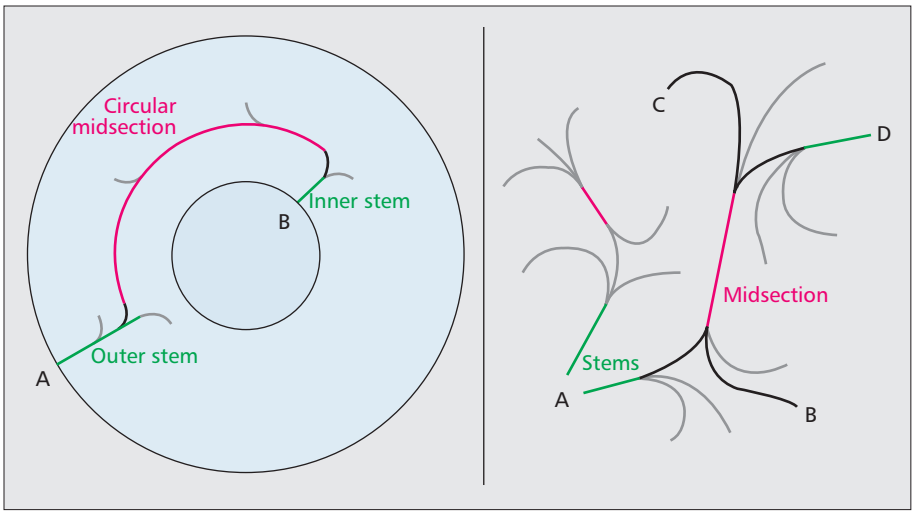

Figure 3. *Left: the three parts of a link in a radial layout of a bipartite graph. Right: more generally, in network diagrams that use edge bundling, we can distinguish between analogous parts of each link.*

tWheel interface, the "where" question can be answered in terms of either source address or destination address (e.g., Fig. 5 has two radial overviews, the left showing sources and the right showing destinations), or in terms of both source and destination in our detailed list view.

Another design principle for visualization is "Overview first, zoom and filter, then details-on-demand" [14]. The radial overview in each subwindow of AlertWheel can be zoomed and rotated by the user, making labels and small details more visible. The pie slices at the center can be expanded, filters can be applied, rolling over a node causes a datatip to appear to provide more details, the user may select a node to fade out other links, and the detailed list view provides details about each alert. In addition, the user may double-click on an alert in the detailed list view to pop up a dialog box containing the raw packet information.

AlertWheel's features also support real analyst tasks. Previous work [6] gives an overview of how analysts work in the real world, describing how analysts distill a dataset in stages, first beginning with the *raw data*, then extracting from that all *interesting activity*, then from that extracting *suspicious activity*, then from that *events*, from that *incidents*, and from that *intrusion sets*. Each subsequent stage results in a smaller subset of data. AlertWheel has built-in support for this workflow of staged drilling down and analysis. As already described, the user may define one or more filters to generate subsets of data. The user may then save a given subset as a scenario. The scenario panel (Fig. 5, bottom right) lists all saved scenarios. When the user creates a scenario (i.e., to save the subset of data currently being viewed), a dialog box pops up in which the user may give the scenario a name, description, and tags, and also selects whether this scenario is considered raw data, interesting activity, suspicious activity, ... or an intrusion set, according to the hierarchy defined by [6]. The user may return to a saved scenario at a later time to review it further. In addition, once one or more scenarios are saved, the filter panel allows the user to create a new filter with a clause saying whether to include or exclude data from one or more existing scenarios, enabling further drill-down and complex queries. Finally, within the detailed list view, the "Status" column indicates whether an individual alert has been saved as part of a scenario.

For example, in Fig. 5, bottom, two events are seen to be marked with a status of "Incident," meaning they were saved within an incident-type scenario. This makes it easier for analysts to review the alerts within the detailed view, to check if all interesting alerts have been recorded within scenarios (and of what type) or if some have been missed.

Although not currently supported, it would be simple to extend AlertWheel to allow searching for scenarios according to keywords or tags, and to share scenarios collaboratively with other analysts, which are other important needs described by [6].

Case Study

We have worked with data from three research honeypot machines. Honeypots are machines made to appear to the outside world as part of a network (e.g., inside an enterprise network), but actually safely isolated, and used as bait to monitor and study the activity of attackers. Our honeypots have gathered much data about real-world attacks that were attempted on them. We are interested in understanding how malevolent software like viruses, worms, and botnets operate and propagate over the Internet. In the former case, we would like to understand the infection scenarios used by these malwares. In the latter case, we would like to see how many machines are infected and their geographic distribution. We now show how AlertWheel can be used to analyze honeypot data.

An attack can usually be divided into five stages: reconnais-

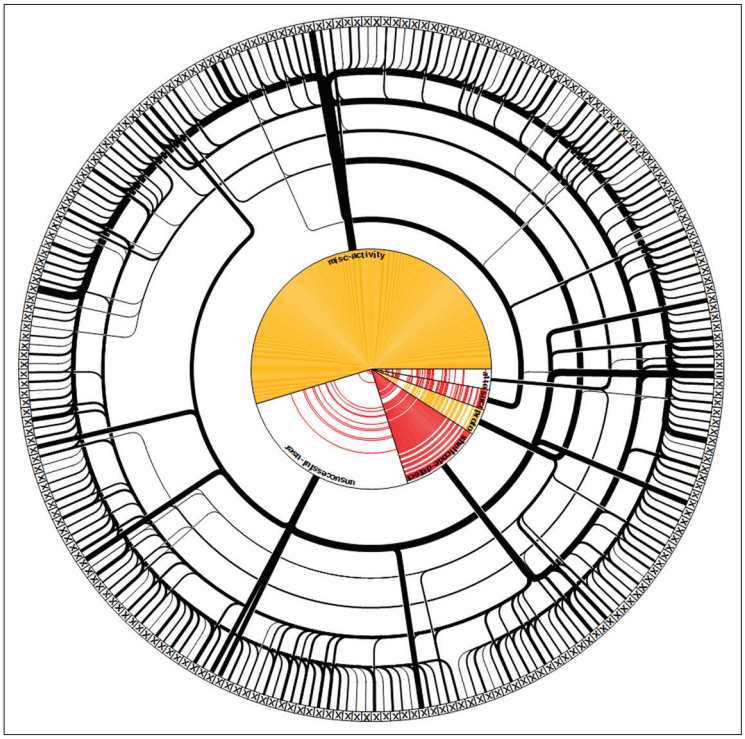

Figure 4. *The same data as in Fig. 1, but with no grouping of the 228 AS nodes (i.e., no neighborhood bundling).*

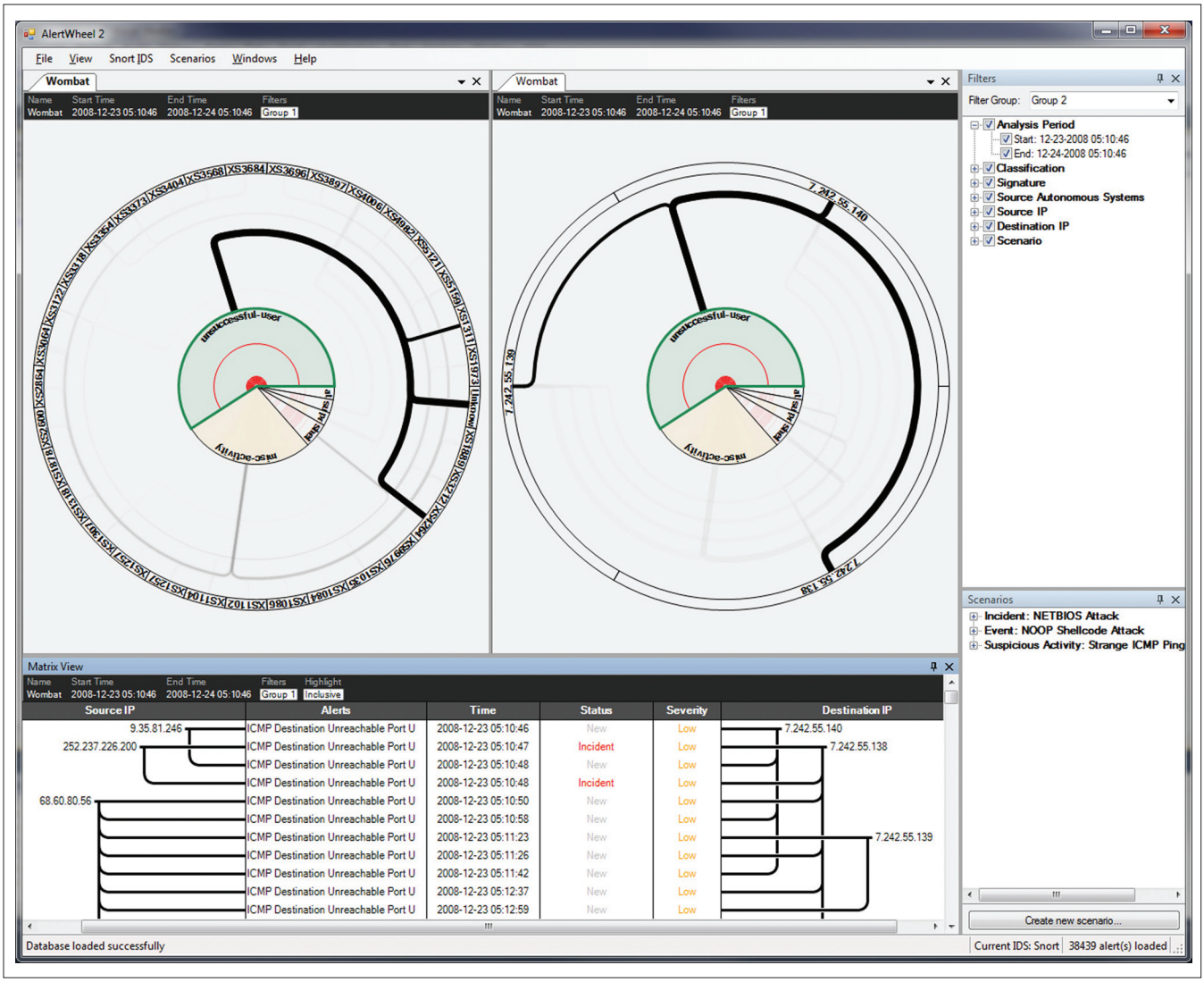

Figure 5. *Three subwindows. The upper left subwindow shows source AS nodes on the outside of the radial layout; the upper right subwindow shows the same data but with the destination IP address shown on the outside of the radial layout. Together, these two views allow the user to simultaneously see the source, category, and destination information for the alerts. The bottom subwindow shows the detailed list view of the same data.*

sance, probing, attack, dig-in, and migration. The first stage involves scanning the network for potential targets. For example, ranges of IP addresses might be scanned with simple PING requests. Once an attacker receives a response, they know a system is located at the given address, and next try to learn more about it. In this second stage, the attacker generally probes the machine for a vulnerable service that could be exploited, allowing them to access data on the machine or take control of it. To do this, the attacker may execute several (valid or invalid) requests. Once a vulnerable service is discovered, the third stage involves launching the actual attack. If the attacker succeeds in exploiting the vulnerable service, the attacker usually attempts to take control of the machine (dig-in), and might then try to migrate to neighboring systems.

In the case of honeypot data, it is likely that the IDS alerts correspond to the first three stages of an attack. It is not surprising, then, that most alerts are classified in the "misc-activity" category (Fig. 1), since this category includes PING requests.

Although a PING alert does not imply an attack, it is suspicious on a honeypot, since outsiders should have no legitimate reason to PING a honeypot. Nevertheless, many analysts ignore PING requests at the start of their analysis. With AlertWheel, we can filter out all PING requests by first applying a filter that removes alerts in the "misc-activity" category.

Doing this with a sample of the honeypot data reveals many severe-level alerts originating from the AS node XS40066 (Fig. 6, top left). Applying an additional filter to only see alerts from this address (Fig. 6, bottom) reveals bands of activity within the pie slices at the center, indicating clear temporal periods of attack, especially from the outside address 68.60.232.17 (selected in green in Fig. 6, bottom left).

Analyzing the corresponding data in the details view (Fig. 6, bottom), a recurring pattern of attack becomes apparent. The attacker is exploiting a known vulnerability of the RPC DCOM service in Windows XP/NT/2000/2003, detected by the Snort IDS as "NETBIOS DCERPC NCACN-IP-TCP ISystemActivator RemoteCreateInstance attempt" (listed in the details view at the bottom of Fig. 6). The attacker simply sends a malformed request on an RPC port of the server, causing a buffer overflow. The attacker then sends shell commands (shellcode), which can then execute operations with administrator privileges. The subsequent alert "ATTACK-RESPONSES Microsoft cmd.exe banner" shows that the

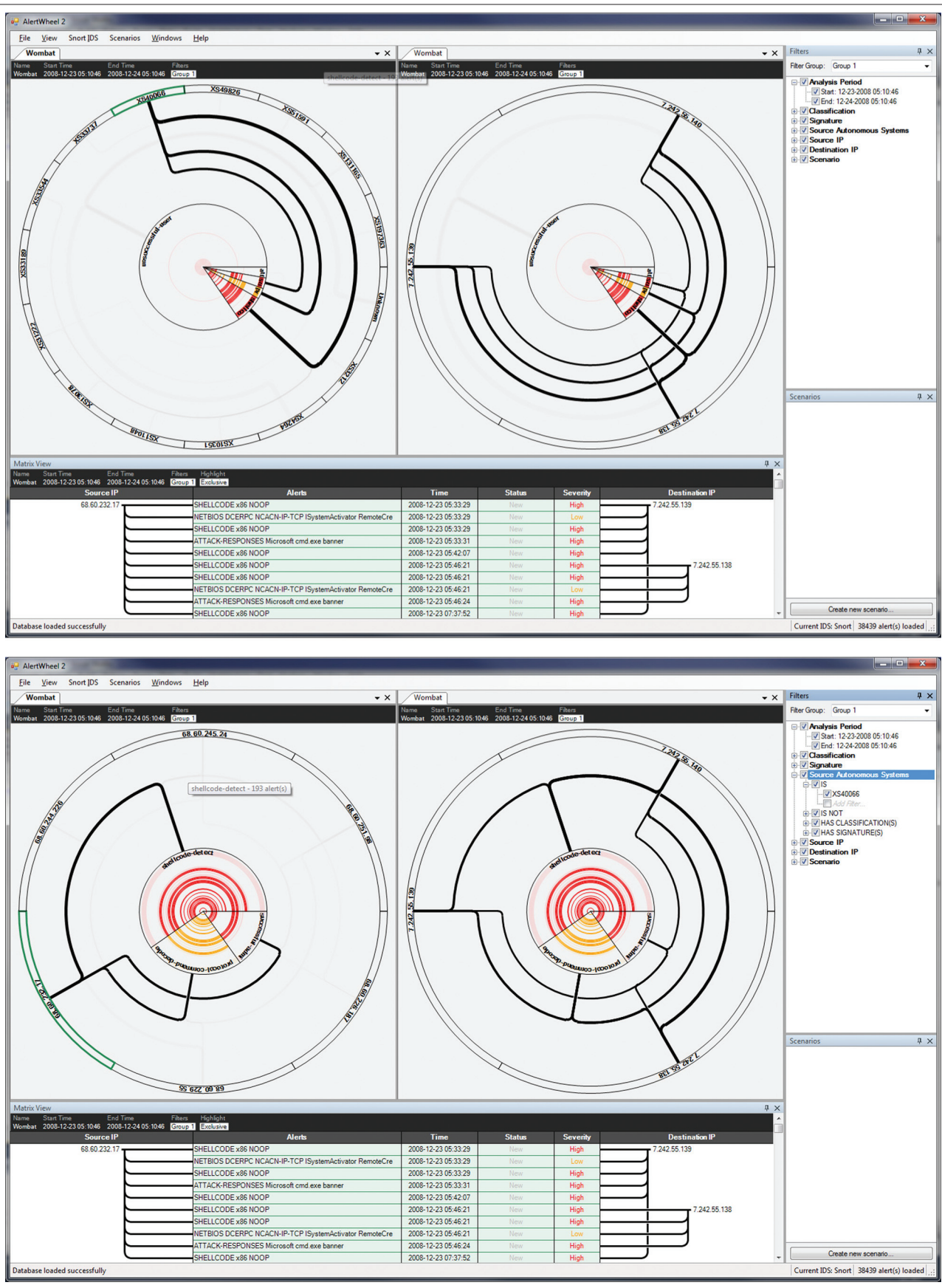

Figure 6. *Two stages in the case study described in the text. The radial overviews at left show source nodes on the outside, and the radial overviews on the right show the three destination(honeypot) machines on the outside. Top: the user has selected the XS40066 AS node, shown in green. Bottom: a filter shows only the IP source addresses for XS40066, and one in particular, 68.60.232.17, is selected in green.*

attack succeeded, and the attacker accessed the Windows shell (this success was actually only simulated by the honeypot). It is likely that the attacker attempted this attack several times, possibly varying the attack each time, perhaps because they were unsatisfied with the responses from the server. It is possible to test this hypothesis by accessing the raw details about alert packets by double-clicking on an alert in the details view.

The foregoing case study illustrates the ease with which the user can filter and analyze almost 40,000 alerts on a single screen, and rapidly detect concrete attacks. Furthermore, this example only used a fraction of AlertWheel's functionality, which allows for great flexibility in the way filters can be defined and in the way views can be combined.

Two stages in the case study described in the text. The radial overviews at left show source nodes on the outside, and the radial overviews on the right show the three destination (honeypot) machines on the outside. Top: the user has selected the XS40066 AS node, shown in green. Bottom: a filter shows only the IP source addresses for XS40066, and one in particular, 68.60.232.17, is selected in green.

Conclusions

AlertWheel incorporates multiple coordinated views showing both overviews and details of IDS data, allowing for zooming, expansion, filtering, and saving of subsets with user-defined annotations. Our user interface follows previous design principles [3, 14] and supports a realistic workflow [6].

We have also introduced news ways of drawing bipartite graphs (Fig. 2) that are applied in both our radial overview and detailed list view, and in the case of the radial overview we combine three bundling techniques without introducing any ambiguity, which is unique in the literature. These methods of displaying bipartite graphs could, of course, be applied outside network security.

The case study presented shows how AlertWheel can analyze attacks on honeypots from the Internet. However, AlertWheel could also be used to visualize IDS alerts generated inside a corporate intranet. In such a case, the outer circle would represent the addresses of infected computers in the enterprise that are generating malicious traffic toward any local or distant target. AlertWheel could also initially show the bipartite graph of outside address × attack type, and then transition to showing the bipartite graphs for outside address × internal address or internal address × attack type in order to investigate attacks against a whole corporate network. Such approaches would eventually have to be evaluated with experienced network analysts.

Acknowledgments

An enthusiastic thank you goes out to Marie-Claire Willig for help setting up our honeypot and programming certain features of AlertWheel. Thanks also to Christophe Viau for helpful comments. This work was funded by NSERC.

References

[1] H. Koike and K. Ohno, "SnortView: Visualization System of Snort Logs," *Proc. ACM Wksp. Visualization and Data Mining for Computer Security*, 2004, pp. 143–47.
[2] K. Abdullah *et al.*, "IDS Rainstorm: Visualizing IDS alarms," *Proc. IEEE Wksp. Visualization for Computer Security*, 2005, pp. 1–10.
[3] S. Foresti *et al.*, "Visual Correlation of Network Alerts," *IEEE Computer Graphics and Applications*, vol. 26, no. 2, Mar./Apr. 2006, pp. 48–59.
[4] G. M. Draper, Y. Livnat, and R. F. Riesenfeld, "A Survey of Radial Methods for Information Visualization," *IEEE Trans. Visualization and Computer Graphics*, vol. 15, no. 5, Sept./Oct. 2009, pp. 759–76.
[5] N. Provos, "A Virtual Honeypot Framework," *Proc. 13th USENIX Security Symp.*, 2004, pp. 1–14.
[6] A. D'Amico and K. Whitley, "The Real Work of Computer Network Defense Analysts," *Proc. Wksp. Visualization for Computer Security*, 2007, pp. 19–37.
[7] N. Henry, J.-D. Fekete, and M. J. McGuffin, "NodeTrix: A Hybrid Visualization of Social Networks," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 6, Nov./Dec. 2007, pp. 1302–09.
[8] D. Holten and J. J. van Wijk, "Force-Directed Edge Bundling for Graph Visualization," *Computer Graphics Forum – EuroVis '09*, vol. 28, no. 3, 2009, pp. 983–90.
[9] S.-J. Luo *et al.*, "Ambiguity-Free Edge-Bundling for Interactive Graph Visualization," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 5, May 2012.
[10] S. Pupyrev, L. Nachmanson, and M. Kaufmann, "Improving Layered Graph Layouts with Edge Bundling," *Proc. 18th Int'l. Symp. Graph Drawing*, ser. LNCS, vol. 6502, 2010, pp. 329–40.
[11] M. Dickerson *et al.*, "Confluent Drawings: Visualizing Non-Planar Diagrams in A Planar Way," *Proc. 11th Int'l. Symp. Graph Drawing*, ser. LNCS, vol. 2912, 2003, pp. 1–12.
[12] F. van Ham, M. Wattenberg, and F. B. Viégas, "Mapping Text with Phrase Nets," *IEEE Trans. Visualization and Computer Graphics*, vol. 15, no. 6, Nov./Dec. 2009, pp. 1169–76.
[13] M. Dumas *et al.*, "Optimizing A Radial Layout of Bipartite Graphs for A Tool Visualizing Security Alerts," *Proc. 19th Int'l. Symp. Graph Drawing*, ser. LNCS, vol. 7034, 2011, pp. 203–14.
[14] B. Shneiderman, "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations," *Proc. IEEE Symp. Visual Languages*, 1996, pp. 336–43.

Biographies

MAXIME DUMAS (maxime.dumas.1@ens.etsmtl.ca) received his Master's degree from ETS in 2011 for his project AlertWheel. His research interests include information visualization, computer security, and financial engineering. He is currently working on his Ph.D. in financial data visualization.

JEAN-MARC ROBERT (jean-marc.robert@etsmtl.ca) has been an associate professor at ETS since June 2006, following 10 years in the telecommunications industry. His research interests focus on the security of telecommunication infrastructures and routing protocols. He has published numerous papers in international conferences and academic journals, and is the author or co-author of 13 U.S. patents.

MICHAEL MCGUFFIN (michael.mcguffin@etsmtl.ca) is an associate professor at ETS, specializing in information visualization and human-computer interaction. He has published three papers cited more than 100 times each, and received an Honorable Mention for his paper at IEEE InfoVis 2009.