Towards Metric-Driven, Application-Specific Visualization of Attack Graphs

Mickael Emirkanian-Bouchard and Lingyu Wang(✉)

Concordia Institute for Information Systems Engineering,
Concordia University, Montreal, Canada
wang@ciise.concordia.ca

Abstract. As a model of vulnerability information, attack graph has seen successes in many automated analyses for defending computer networks against potential intrusions. On the other hand, attack graph has long been criticized for the lack of scalability when serving as a visualization model for conveying vulnerability information to human analysts. In this paper, we propose two novel approaches to improving attack graph visualization. First, we employ recent advances in network security metrics to design metric-driven visualization techniques, which render the most critical information the most visible. Second, existing techniques usually aim at an one-size-fits-all solution, which actually renders them less effective for specific applications, and hence we propose to design application-specific visualization solutions for network overview and situational awareness. We discuss the models, algorithms, implementation, and simulation results.

1 Introduction

Computer networks have long become the nerve system of enterprise information systems and critical infrastructures. On the other hand, the scale and severity of security threats to computer networks have continued to grow at an ever-increasing pace. To defend computer networks against potential attacks, an important starting point is to understand the networks' weaknesses and flaws. To that end, a network security administrator or analyst should be capable of assessing the security posture of a network quickly and efficiently. However, the amount of vulnerability information in a network increases quickly in the network's size, mostly because vulnerabilities are seldom independent and attackers may combine them in sophisticated ways for attack propagation or privilege escalation. Therefore, conveying a large amount of vulnerability information to human analysts is a challenging issue for most networks.

Attack graph is an established model of vulnerability information in networks [1,22]. By encoding potential exploits of vulnerabilities and linking them through their common pre- and post-conditions, an attack graph provides a clear picture about how attackers may potentially break into a network and subsequently compromise network assets. Attack graphs have seen successes in many automated analyses for assessing, monitoring, and hardening computer networks.

On the other hand, attack graph has long been criticized for its poor scalability when serving as a visualization model for human analysts to comprehend, since even a small network may yield an attack graph that is too complex to understand [17].

The visualization of attack graphs has received limited attention (a more detailed review of related work will be given in Sect. 5). The scalability may be partially improved by abstracting and hiding low-level details [17], although the improvement is often limited since the method still relies on the same node-link representation of attack graphs. The clustered adjacency matrices [18] address the scalability issues but lead to a highly abstract model unsuitable for human interpretation. GARNET [24] and NAVIGATOR [6] employ tree-based structures to represent host configuration, but they both lack sufficient details about connectivity and exploit relationships.

In this paper, we propose two novel approaches to improving attack graph visualization. First, we employ recent advances in network security metrics to design *metric-driven visualization* techniques. Such techniques prioritize the visualization based on relative metric scores. This will allow the most critical information to be best highlighted or magnified in order to guide human analysts to explore the most pertinent threats. Second, we observe that most existing attack graph visualization techniques aim at an one-size-fits-all solution, which actually renders them less effective for specific applications; we then propose to design *application-specific visualization* solutions. In this paper, we focus on two such solutions, namely, the *radial attack treemaps* for network overview and the *topographic attack trees* for situational awareness. We discuss models, algorithms, implementation, and simulation results.

The rest of this paper is organized as follows. Section 2 reviews background information on attack graph, security metrics, and relevant visualization techniques. We will then introduce two novel attack graph visualization models for network overview and situational awareness in Sects. 3 and 4, respectively. Finally, Sect. 5 reviews related work and Sect. 6 concludes the paper.

2 Preliminaries

To be self-contained, this section reviews background information on attack graph, security metrics and visualization techniques.

2.1 Attack Graph and the Scalability Issue

Attack graph models vulnerabilities and their inter-dependency inside a network [1,22]. An attack graph can be represented as a directed graph, with exploits and conditions as vertices, and the causal relationships between exploits and conditions as edges.

The left-hand side of Fig. 1 shows our running example which will be used throughout the paper to illustrate different visualization methods. On the right-hand side of the figure is a toy network and on the left side the corresponding attack graph, in which each predicate *vulnerability(source host, destination* *host)* inside an oval indicates a self-explanatory exploit, and each plaintext *condition($host_1$,$host_2$)* or *condition(host)* indicates a security-related condition. Edges point either from an exploit's pre-conditions to the exploit (e.g., a user privilege on host 1 is a pre-condition for exploits originated from host 1), or from the exploit to its post-conditions. Note the numbers inside the attack graph can be ignored for now, and they will be needed in later discussions. More formally,

Definition 1. *An attack graph G is a directed graph $G(E \cup C, R_r \cup R_i)$ where E is a set of exploits, C a set of conditions, $R_r \subseteq C \times E$ the require relation, and $R_i \subseteq E \times C$ the imply relation.*

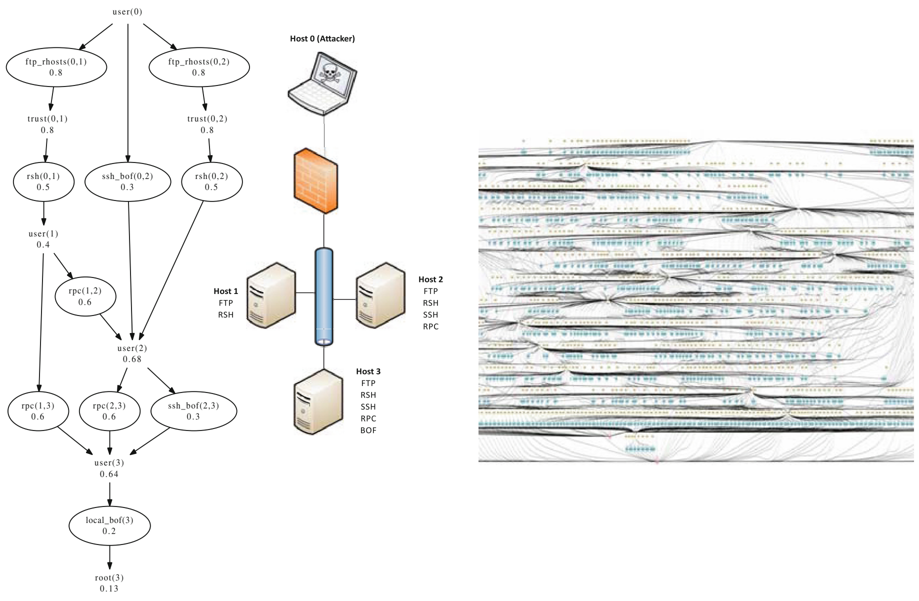


Fig. 1. The running example (Left) and attack graph of a 14-host network (Right)

The above basic representation of attack graphs is more suitable for automated analysis than for visualization-based human analysis. Enumerating all the exploits, their pre- and post-conditions, and edges between them in a single directed graph will inevitably lead to very high node and edge density, a significant amount of crossings between edges, highly complex edge paths, and a high average edge length. These characteristics render the attack graph messy and difficult to comprehend, and prevent human analysts from interpreting the attack graph and cross validating with results of automated analysis. As an example, the right-hand side of Fig. 1 shows a messy and illegible attack graph. It may be surprising to note that this attack graph actually represents a small network composed of only 14 machines, each of which has less than 10 vulnerabilities. Clearly, the basic representation of attack graphs is not a viable visualization solution.

2.2 Security Metrics

Scoring and ranking vulnerabilities and networks based on their relative severity and security has drawn significant attentions. Among existing efforts, the Common Vulnerability Scoring System (CVSS) is a widely recognized standard for security vendors and analysts to assign numerical scores to vulnerabilities to reflect their relative severity [21]. The approach in [8] first assigns a normalized CVSS score as the conditional probability of successfully executing each exploit of the vulnerability given satisfied pre-conditions. The assigned probabilities are then used to build a Bayesian network based on causal relationships between exploits and used to find the probability that critical assets are compromised, which provides a security metric for the whole network. For example, in Fig. 1, a number inside an oval is the aforementioned conditional probability and under each condition is the probability of satisfying that condition.

In this paper, we extend the above Bayesian network-based security metric by introducing the notion of *asset value* to attack graphs, which is a numerical value between 0 and 10 (corresponding to the domain of CVSS scores) assigned by administrators to each condition in the attack graph based on the condition's relative significance with regards to confidentiality, integrity, and availability. From this assigned asset value, we calculate the *risk* at multiple hierarchical levels for conditions, hosts, groups of hosts (subnets), and networks. Here we adopt the common approach of defining risk as the product of the asset value and attack likelihood (that is, the probability obtained using the aforementioned Bayesian network approach). More specifically,

Definition 2. *Given the probability of executing each exploit $P(e)$ and that of satisfying a condition $P(c)$ inside an attack graph $G(E \cup C, R_r \cup R_i)$, and an asset value assignment function $AV(.) : C \to [0, 10]$, we define*

- *the risk of a condition c as $R_c(c) = \frac{P(c) * AV(h)}{10}$.*
- *the risk of a host h as $R_h(h) = R_c(< root, h >)$.*
- *the risk of a group of hosts (or the whole network) G as $R_g(G) = \sum_{h \in G} R_h(h)$.*

2.3 Applying Existing Visualization Models

We apply several existing visualization models to attack graph to demonstrate their limitations and motivate further discussions.

Balloon Attack Graph. Due to the hierarchical nature of most networks, an obvious approach for improving the scalability of attack graphs is to grouping or clustering certain nodes which share similar characteristics (e.g., residing on the same or adjacent hosts [17]). However, such an approach will meet difficulty to maintain readability without losing valuable information due to the relatively high edge density and crossings in a usually highly-connected attack graph.

In Fig. 2, we apply to our running example a clustering method with multiple cluster centers in order to form clusters of nodes without a pre-defined top-down path or a particular directional layout, based on the clustering method proposed by Melancon et al. [16] which aims to achieve a balanced layout, namely, a *balloon attack graph.*

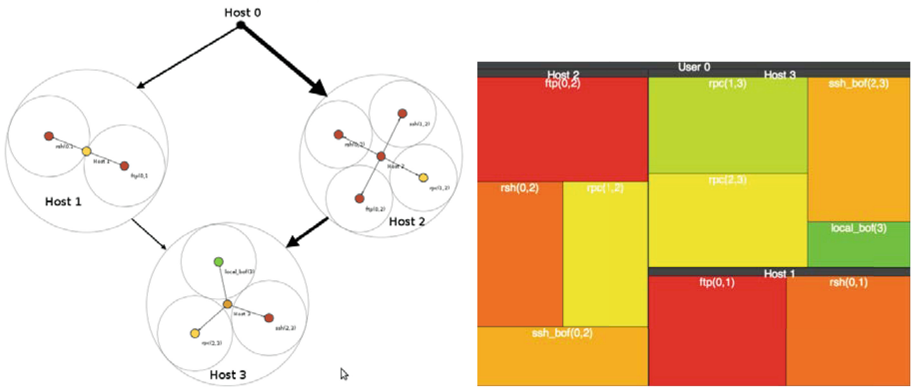


Fig. 2. Balloon attack graph and attack treemap

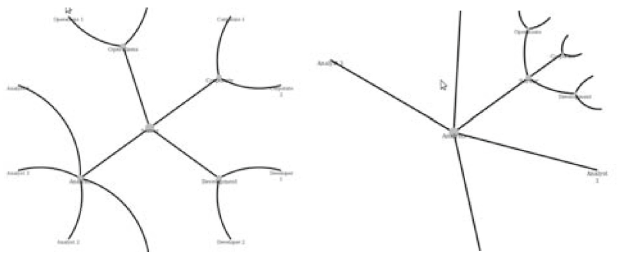
From the example, it is clear that this visualization model can improve the density of nodes as well as the readability to some extent, through clustering exploits associated to the same host. However, it is equally clear that the edges cannot be fully displayed (without breaking the balloons), leading to a significant loss of information; the improvement of scalability is also quite limited.

Attack Treemap. An issue with conventional node-edge attack graph is the difficulty of expressing the hierarchical relationships between exploits, hosts, and networks. The above balloon attack graph addresses this through clustering nodes into balloons, but it also wastes much visualization space to explicitly depict the hierarchical relationships.

To that end, treemaps allow for implicit representation of hierarchical information inside a rectangular display, where the entirety of the visualization space is put to use [11]. Figure 2 shows an *attack Treemap* using our running example, built with the JavaScript InfoVis Toolkit [3] using the binary tiling algorithm [23]. In the attack treemap, each rectangle with a black bar at the top represents a host, inside which each colored rectangle represents an exploit. The color denotes the CVSS score, and the relative size of rectangles denotes the risk value as calculated before.

Clearly, treemap is a dense and relatively scalable visualization model. In addition, GARNET [24] has shown how to add reachability results to treemaps by interactively displaying them through semantic substrates. However, most of the connectivity information and edges in attack graphs are still missing here, and adding them as overlying edges will clearly lead to a messy result. We will address this issue in Sect. 3.

Hyperbolic Attack Tree. As attack graphs get larger, screen size becomes a concern, and forcing an analyst to zoom or pan on sections of an attack graph will likely lead to a loss of context or awareness of the overall network. To this end, the hyperbolic geometry offers opportunities for creating a fisheye-lens effect, with the center of the graph (the focus) occupying the most space and the remainder of the graph condensed and pushed outwards, which helps to maintain the context and awareness of the whole graph [2]. Figure 3 shows a *hyperbolic attack tree* based on our running example.



(a) A Hyperbolic Tree (b) A Re-Centered Tree

Fig. 3. Hyperbolic attack tree

The constant contextual awareness makes hyperbolic attack trees an appealing choice for applications like situational awareness. We will revisit this approach in Sect. 4.

3 Radial Attack Treemaps

This section introduces a scalable, metric-driven visualization model, the *radial attack treemap*, for the purpose of obtaining a quick overview of a network's vulnerability information. We first give an overview, followed by the description of models and algorithms, and finally we present simulation results.

3.1 Overview

Enabling a security analyst to acquire a quick overview of the entire network's vulnerability information is a key tactical advantage in assessing networks' security. The goal here is to encode as much legible details as possible inside a given size canvas. Section 2.3 mentioned treemaps as a visualization model that provides relatively high information density and scalability by occupying the entirety of the canvas. On the other hand, the main shortcoming of treemaps lies in the difficulty of displaying edges between exploits.

Intuitively speaking, our main idea here is to *bend the treemap into a ring, and display edges inside that ring.* As to the actual display of edges, we turn to radial graphs, which allows a fixed-size layout with high information density, element proximity, and edge management [14]. Unlike conventional graphs in which an edge may be obstructed by a node, in a radial graph, a line between two points on a circle is an unobstructed line. Moreover, the edges in a radial graph can be hierarchically bundled with crossings between edges minimized.

By combining key concepts of treemaps and radial graphs, we propose a metric-driven and treemap-based radial visualization, namely, the *radial attack treemap*. We summarize the key features and advantages of this novel visualization model in the following, while leaving details of the model and implementation to later sub-sections:

– The model provides a quick overview of exploits, chains of exploits (that is, paths in an attack graph), hosts, and causal relationships between exploits in a network.
– The color and size of each slice of the outside ring represents the CVSS score and risk of the corresponding exploit, respectively.
– The stacking of slices and sub-slices in the outside ring implicitly represent hierarchical relationships between exploits, exploit chains, and hosts, reducing the number of edges that need to be explicitly displayed (in contrast to the original attack graph).
– The center of the ring displays edges in a bundled way to minimize the number of crossings between edges, leading to a cleaner visualization result.
– Layout of the bent treemaps is optimized such that the lower level details are displayed more towards the outer side of the ring in order to occupy more space.

Figure 4 illustrates an example of radial attack treemap, which is based on our running example shown in Fig. 1.

3.2 Models and Algorithms

Definition 3 more precisely describes the radial attack treemap.

Definition 3 (Radial Attack Treemap). *Given an attack graph $G(E \cup C, R_r \cup R_i)$ with hosts H and the risk function R_c, R_h, and R_g, a radial attack treemap is composed of a ring R and a collection of links L, where*

– *R is divided into a collection of slices S, with each slice $s \in S$ corresponding to a host $h \in H$.*
– *each slice s is divided into a collection of subslices SS, with each subslice $ss \in SS$ corresponding to an exploit chain (a sequence of exploits involving the destination host h, the same source host, and leading to $< root, h >$.*
– *each subslice ss is further divided into a collection of subsubslices SSS, in which each subsubslice $sss \in SSS$ corresponds to an exploit e in the exploit chain.*

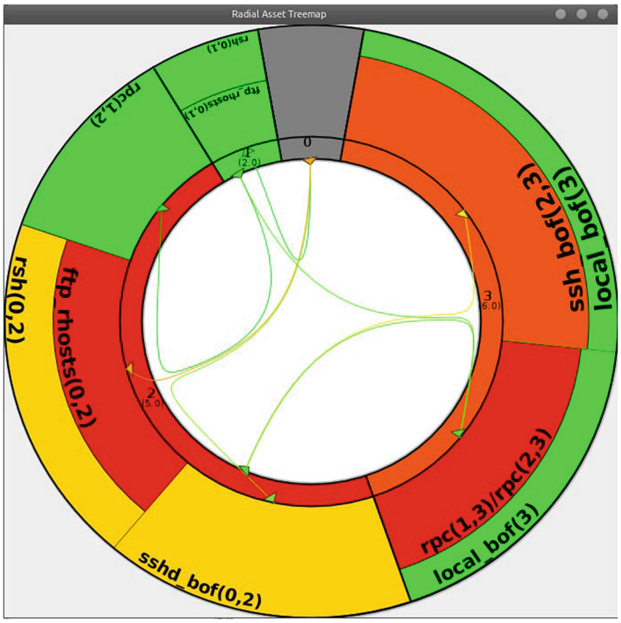


Fig. 4. Radial attack treemap

- *the relative size of each slice, subslice, and subsubslice is proportional to the risk score (Definition 2) of corresponding host, exploit chain, and exploit, respectively (details will be provided later).*
- *the color of each subsubslice represents the CVSS score of the corresponding exploit (details will be provided later).*
- *each link in L points from a slice corresponding to host h, to a subslice corresponding to an exploit chain involving the source host h.*
- *all the links in L are bundled and routed through the center of the ring R.*

Data Structures. We now describe the data structures required for implementing the proposed visualization model. Specifically, to implement the model, we need to compute the aforementioned risk metrics and convert a given attack graph into a suitable data structure. We then derive geometric information necessary to the final rendering of the model. Therefore, for each element in the model, there will be a corresponding view element containing additional information necessary to the visualization, as detailed below.

- *Exploit & Subsubslice:* Each exploit is a list of five attributes, an identifier, a set of pre-conditions and post-conditions, a CVSS score, and a risk value. Correspondingly, a subsubslice, as the view representation of the exploit, is a list of attributes including a label, a color derived from a normalized CVSS score, as well as a size proportional to the risk value of exploit chain.

- *Exploit Chain & Subslice:* Each exploit chain is a list of attributes including an identifier, a risk value, as well as the source host involved. Correspondingly, a subslice is a list of attributes including references to the composing subsubslices, a label, a size derived from the risk value, an anchor point which is a set of coordinates used as destination points for incoming links, and a color derived from the CVSS scores of the corresponding exploits.
- *Host & Slice:* A host is a list of attributes including the references to the composing exploit chains, an identifier, and a risk value. Correspondingly, a slice is a list of attributes including the host name, references to the composing subslices, a label, a color derived from the CVSS scores, a size derived from the risk value of the host, and two anchor points, with the first being a set of coordinates used as intermediate destination points for incoming links and the second being a set of coordinates used as the source points for outgoing links from this host.
- *Link:* A link is a pair $< h, ec >$ indicating the source host h involved by exploits in the exploit chain ec. Correspondingly, the link is visualized using the Bézier spline composed of two curves, a cubic Bézier curve and a quadratic Bézier curve [20]. The former contains three sets of coordinates, namely, a start point, an end point and a control point, while the latter has four, namely, a start point, an end point and two control points.

Algorithms. This subsection discusses two series of algorithms. The first converts a given attack graph to the data structures mentioned in the previous sub-section. The second is for computing geometric information used in creating the view structures.

First, in the following, Algorithm 1 uses a recursive depth-first search in the input attack graph to obtain all paths from user-access conditions to the root condition of the target host (Algorithm 2). For each path obtained, we verify that all exploit sequences leading to this condition have all their pre-conditions satisfied and that the path generated is valid (detailed algorithm is omitted due to space limitations).

Second, we discuss how the view data structures may be generated (detailed algorithms are omitted due to space limitations). The ring is generated by converting exploits, exploit chains and hosts into subsubslices, subslices and slices, respectively. Host and exploit chain risk scores are expressed by the angle of ring

Algorithm 1. getAllExploitChains

```
Input: An attack graph, a set of host-access conditions $Host$
Output: A set of Hosts possessing exploit chains and exploits
1 foreach $Host\ to \in Hosts$ do
2     foreach $Host\ from \in Hosts$ do
3         $paths_{from->to}[\ ][\ ] \leftarrow getAllPaths(from, to)$;
4         foreach $path\ p \in paths_{from->to}$ do
5             if $isValid(true, path, from, initialconditions)$ then
6                 $to.addExploitChain(path)$;
```

Algorithm 2. getAllPaths

```
Input: A Linked List of visited nodes $visited$, the end condition $end$
$Node\ n = visited.last()$;
$Node[]\ nodes = n.getNexts()$;
foreach $Node\ n \in Nodes$ do
    if $visited.contains(node)$ then
        $continue$;
    $visited.add(n)$;
    $Node[]\ path \leftarrow visited$;
    $allPaths.add(path)$;
    $visited.removeLast()$;
foreach $Node\ n \in Nodes$ do
    if $visited.contains(n)\ ||\ n = end$ then
        $continue$;
    $visited.addLast(n)$;
    $getPath(visited, end)$;
    $visited.removeLast()$;
```

segments they occupy. $Host_0$, representing the initial attacker-controlled host, possesses a fixed angle, α_0. The slices representing a given host x will have an angle α_x of value:

$$\alpha_x = (360 - \alpha_0) * \frac{score_x}{\sum_{i=1}^{n} score_i} \tag{1}$$

Similarly, the angle α_y of an exploit chain $ec \in h_x$, relative to risk of the other exploit chains of the host – will have a value of:

$$\alpha_y = \alpha_x * \frac{score_{ec}}{\sum_{ec \in h} score_{ec}} \tag{2}$$

For an exploit $e \in ec$, the angle of ring segment it occupies is the same as that by its exploit chain parent, and occupied area thus depends on the length of the radius segment between the current exploit and the next exploit (or the ring's two edges), depending on the risk scores of these exploits' post-conditions. The color of subsubslice is derived from the normalized CVSS scores of the vulnerabilities using a *color ramping algorithm* similar to the one described by Bourke in [4].

The links displayed at the center are Bézier splines [20] computed using the method by Holten in [9]. The spline is composed of two Bézier curves, the *origin curve*, a cubic Bézier curve with two control points starting at the origin host's anchor point denoted by point 0 and ending on the destination host's projection on the host circle, and the *destination curve*, a quadratic Bézier curve starting at the end of the origin curve and ending at the exploit chain anchor point. Finally, the link color is derived from the score of the first exploit in the exploit chain of the destination, using a color ramping algorithm.

3.3 Implementation and Simulation

A prototype was built using Java and the Graphics2D and Curve2D libraries, included in the JavaSE package. It is built using the Model-View Controller [12] (MVC) pattern. A GraphViz [7] .dot file parser reads an input attack graph and loads it into memory. The graph is then traversed to generate exploits, exploit chains, and hosts, using Algorithms 1 and 2. This model is then converted into slices, subslices, subsubslices, and links, in order to generate the ring and links.

We now study the density and scalability of the visualization model through simulation using randomly generated attack graphs (we note that although an experiment using real world data is certainly more desirable, to the best of our knowledge, a publicly available dataset containing a significant number of attack graphs is not currently available). We generate 1200 attack graphs using Python programs from small seed graphs based on real world attack graphs. The simulation environment is a dual-core Intel Core i5 processor with 8 GB of RAM running Debian 7. The entire application was written in Java and runs on OpenJDK 6.

We compare the scalability of radial attack treemaps with that of the input attack graphs. As a radial asset treemap is designed as a fixed-size visualization, we set a threshold value for the smallest allowable subsubslice, at $1000px^2$ (leaving approximately 10 characters at 8pt. font size), and we ensure all subsubslices in a radial attack treemap to be legible by scaling them according to this threshold. Figure 5 shows the average canvas size of both models in relation to the number of hosts.

The simulation clearly confirms that radial attack treemaps offer a higher information density than conventional attack graphs. Figure 5 shows that, on average, 10 hosts can be represented on a canvas of merely 900×900 pixels, while the corresponding attack graphs would require a canvas of over 2800×2800 pixels.

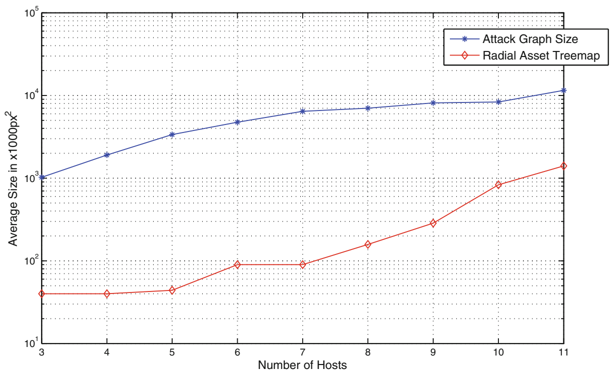


Fig. 5. The Comparison of average sizes

Next, we study the degree of reduction in the number of edges/links by implicitly representing edges in the radial attack treemaps (through stacking subsubslices). We note that, in addition to this reduction in the number of edges/links, the radial attack tree maps have other advantages in terms of displaying links, as mentioned already in the previous subsection. Figure 6 compares the number of edges/links in relation to the number of hosts in the graph.

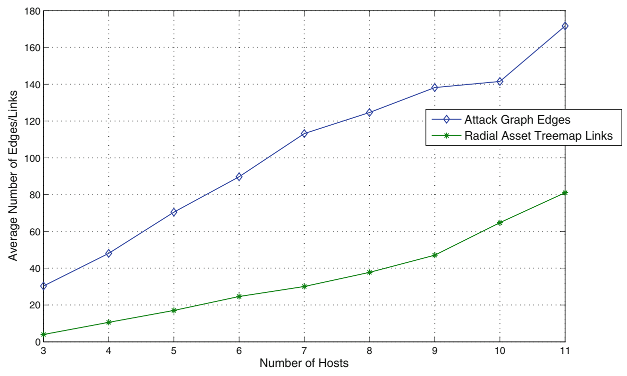


Fig. 6. Average number of Edges/Links

This simulation indicates that the amount of edges/links has been reduced to approximately a third those of conventional attack graphs. The implicit relationships between host, exploit chains and exploits allow for such a significant edge reduction.

We note that, Fig. 5 seems to indicate that the rate of growth of radial asset treemap is greater than that of conventional attack graphs. This is a consequence of the ring's tiling algorithm: regardless of the size of the canvas, an exploit chain's partition angle will remain the same. When compared to a two-dimensional graph canvas, both size increases are quadratic but with the partition size depending on the angle of its parent exploit chain, leading to a lower rate of growth of partition surface compared to the available surface of a rectangular canvas.

3.4 Discussions

The proposed metric-driven radial attack treemap provides a viable visualization solution for human analysts to quickly grasp an overview of the vulnerability information in a network. Nonetheless, the model in its current form still has a few limitations. First, due to the limited level of hierarchy in the treemaps, it will be difficult to visualize a large network in a single view. Developing a new tiling algorithm to support more levels of hierarchy or using interactivity to vary the hierarchy levels of slices or filtering out certain nodes are both viable solutions. How well those solutions would scale to larger networks with hundreds of hosts will need to be confirmed through experiments. Second, the trade-off between the areas occupied by the ring and by links requires developing algorithms to optimize such a trade-off for clarify on both sides.

4 Topographic Hyperbolic Trees

This section introduces the novel *topographic hyperbolic tree* model for monitoring and predicting real time progress of attacks.

4.1 Overview

One important aspect of visualization in the application of cyber-situational awareness is to allow administrators to see both the current focus of an ongoing attack and most likely next steps. Another important aspect is to provide a sense of *distance* between potential attack steps based on the number of intermediate steps or relative difficulty of such steps [24,25]. In Sect. 2.3, we have shown that the hyperbolic tree model is a suitable model for the first purpose. As to express the attack distance, we are inspired by geographical topographic maps, in which contour lines are used to indicate fixed increases in altitude. Therefore, the main idea here is to *enhance the hyperbolic attack tree model with contour lines representing attack steps at similar distance.* Again, we summarize the key features and advantages of this novel visualization model in the following, while leaving details of the model and implementation to later sub-sections:

- It provides an interactive visualization of ongoing attacks and plausible next steps.
- The hyperbolic tree creates a fisheye-lens effect that allows administrators to focus on the current attack and its closest future steps, while not losing context or awareness of other steps that may be further away but are still possible, such as the ultimate goal of the attack.
- The contour lines provide a rough idea about future attack steps that are at similar distance from the current step.
- In addition, the relative length of different edges represent (after taking into account the fisheye-lens effect) the relative difficulty of the corresponding exploit.

Figure 7 shows the topographic hyperbolic tree for our running example.

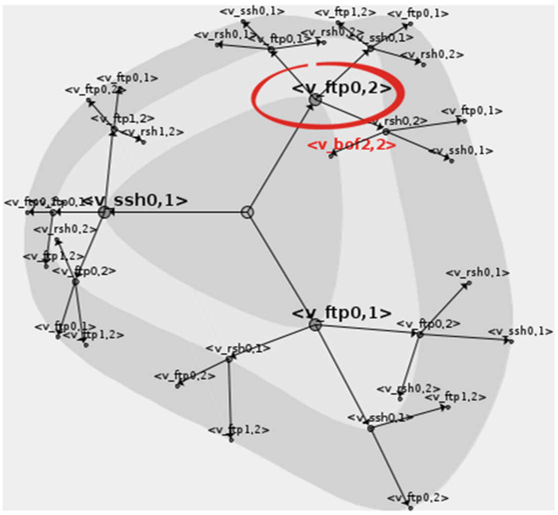


Fig. 7. Topographic hyperbolic tree

4.2 Model and Algorithms

Definition 4 formally describes the topographic hyperbolic tree.

Definition 4. *Given an attack graph $G(E \cup C, R_r \cup R_i)$ with hosts H and the risk function R_c, R_h, and R_g, a topographic hyperbolic tree is composed of a hyperbolic attack tree $T(E, C)$, which has the exploits E as nodes and conditions C as edges, and a collection of contour lines L linking all the exploits sharing the same depth in the tree. The relative length of an edge is based on the risk metric score of the corresponding condition as well as the depth of the node (more details will be provided later).*

The construction of a topographic hyperbolic tree from an input attack graph involves a few steps. We first load the attack graph into memory. Then, for each time the graph is recentered, we apply the tree generation algorithms. We then layout the tree on the canvas and generate the contour lines. More specifically (detailed algorithms are omitted due to space limitations),

1. We start by establishing the context required to initiate the graph traversal using (Algorithm 3), and then recursively perform the graph traversal and tree construction using Algorithm 4, while limiting the maximal depth of any tree branch to be a pre-defined parameter MAX_DEPTH in order to avoid the explosion of possible paths.
2. We then layout nodes on the canvas. We compute the coordinates of every node by calculating the length of a link as well as the angle at the origin. The length of an edge is a function of the risk of the pre-conditions of the exploit represented, as well as the number of steps from the center: $distance_{child} = \frac{score_{child}}{c} * (MAX\_DEPTH - step_{child} + 1)$. The angle of a node's children

will depend on the angle of the parent as well as the number of children this parent possesses: $\alpha_c = \frac{180 - c*step}{nbChildren}$.

3. Finally, we generate and draw the contour lines. Three main steps are required for the drawing. First, after obtaining all points at a given level i, we ensure that the polygon formed by these points completely includes the polygon formed by the points at a previous level $i-1$. Otherwise, we add the points of polygon $i-1$ lying outside of polygon i to the polygon i. Second, we ensure that each polygon is convex. If the polygon is concave, we apply a convex-hull algorithm commonly called the Gift-Wrapping Algorithm [10]. Finally, we smooth the lines by interpolating the points using the Catmull-Rom Algorithm [5].

Algorithm 3. The Traversal Initiation Function

```
Input: A tree $tree$, an attack graph $graph$, a list of conditions $attackerKnowledge$
Output: A tree $tree$ representing all possible attacker paths from given initial conditions
$Node[\,]\ firstNextSteps \leftarrow getNextSteps(initialConditions)$;
for $Node\ n \in firstNextSteps$ do
    $knowledge.add(n)$;
    $knowledge.add(n.getNexts())$;
    $traverse(tree, n, knowledge, 1)$;
return $tree$;
```

Algorithm 4. The Tree Generating Algorithm

```
Input: A tree node $previous$, an attack graph node $graphnode$, a list of conditions
       $attackerKnowledge$ a depth $depth$
Output: The fully expanded tree representing all possible attacker paths
$Nodecurrent = graphNode$;
if $previous = FINAL\_CONDITION\ \&\&\ depth \leq MAX\_DEPTH$ then
    $previous.addNext(current)$;
    $current.addPrevious(previous)$;
    $Node[]\ nextSteps = getNextSteps(attackerKnowledge)$;
    for $Node\ n \in nextSteps$ do
        if (! $attackerKnowledge.contains(n.getNexts())$ then
            $attackerKnowledge.add(n.getNext)$;
            $attackerKnowledge.add(n)$;
            return $traverse(current, n, attackerKnowledge, depth + 1)$
```

We have implemented a prototype using Java and the Graphics2D library. Our simulation (detailed results omitted due to space limitations) shows that both the visualization result and the running time are easily manageable with the maximal depth set to about six, whereas unsurprisingly there is a sharp increase in both thereafter. Given the interactive nature of this visualization model, we believe the model is still useful for many practical applications. On the other hand, further study is needed to improve the tree expansion algorithms in order to avoid the exponential explosion, and to find more efficient ways for incrementally updating the model after each centering operation.

5 Related Work

Sheyner *et al.* firstly employ a model checker to generate all possible attack paths for a network, namely, an attack graph [22]. Since such a model-checking technique suffers from scalability issues, the *monotonicity assumption* stating that an attacker never relinquishes a gained privilege, is employed to achieve a polynomial complexity [1], which is further improved by Ou *et al.* in developing the MulVAL tool [19]. Related efforts on security metrics include the Common Vulnerability Scoring System (CVSS) which is a widely recognized standard for scoring and ranking vulnerabilities [21]. Frigault *et al.* [8] convert attack graphs into Bayesian networks to analyze vulnerability metrics using a probabilistic model. As to related efforts on visualization, Treemaps are introduced as a graphical representation of a weighted tree by recursively partitioning rectangles depending on the weight assigned to the node [11]. The shape of the partitions is dictated by *tiling algorithms*, as reviewed in [23]. Hyperbolic trees (or hypertrees) are introduced by Lamping *et al.* [13] as a *focus+context* technique to create a fisheye effect for viewing and manipulating large hierarchies. There has recently been much focus on radial visualization models across different scientific fields, such as VisAware [15], a radial visualization system representing situational awareness in a generalized way, which is further adapted for intrusion detection in VisAlert [14]. Attack graph visualization presents additional challenges due to their specific requirements. Noel *et al.* [17] present a framework for hierarchically aggregating nodes in an attack graph. Noel et al. [18] also make use of clustered adjacency matrices to compute the reachability and distance similar to a heatmap. GARNET [24] is an Attack Graph visualization tool which outputs treemaps with semantic substrates to visualize a network and its reachability. GARNET later evolves into the NAVIGATOR (Network Asset VIsualization: Graphs, ATtacks, Operational Recommendations) [6], with improvements like the possibility of zooming-in to the host level and displaying port numbers and possible exploits on these ports.

6 Conclusion and Future Work

We have proposed two novel approaches to attack graph visualization, namely, metric-driven visualization and application-specific visualization. Specifically, we proposed a new visualization model by combining treemaps with radial graph for the use case of network overview. Second, we enhanced hyperbolic attack trees with contour lines borrowed from topological maps for the purpose of situational awareness. In addition to future work already mentioned in Sects. 3 and 4, we will also pursue metric-driven visualization models for other applications of attack graphs.

Acknowledgements. The authors thank the anonymous reviewers for their valuable comments. This work is partially supported by Natural Science and Engineering Research Council of Canada under Grant N01035.

References

1. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 217–224. ACM (2002)
2. Anderson, J.W.: Hyperbolic Geometry. Springer, New York (2007)
3. Belmonte, N.G.: The JavaScript InfoVis toolkit. http://www.thejit.org. Accessed 2 Mar 2013
4. Bourke, P.: Colour ramping for data visualization. http://local.wasp.uwa.edu.au/pbourke/texture_colour/colourramp/. Accessed 18 Nov 2012
5. Catmull, E., Rom, R.: A class of local interpolating splines. Comput. Aided Geom. Des. **74**, 317–326 (1974)
6. Chu, M., Ingols, K., Lippmann, R., Webster, S., Boyer, S.: Visualizing attack graphs, reachability, and trust relationships with navigator. In: Proceedings of the Seventh International Symposium on Visualization for Cyber Security, pp. 22–33. ACM (2010)
7. Ellson, J., Gansner, E., Koutsofios, L., North, S.C., Woodhull, G.: Graphviz—open source graph drawing tools. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 483–484. Springer, Heidelberg (2002). doi:10.1007/3-540-45848-4_57
8. Frigault, M., Wang, L., Singhal, A., Jajodia, S.: Measuring network security using dynamic Bayesian network. In: Proceedings of the 4th ACM workshop on Quality of protection, QoP 2008, pp. 23–30. ACM, New York (2008)
9. Holten, D.: Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. IEEE Trans. Visual. Comput. Graph. **12**, 741–748 (2006)
10. Jarvis, R.A.: On the identification of the convex hull of a finite set of points in the plane. Inf. Process. Lett. **2**(1), 18–21 (1973)
11. Johnson, B., Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the IEEE Conference on Visualization 1991, pp. 284–291, October 1991
12. Krasner, G.E., Pope, S.T., et al.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. J. Object Oriented Program. **1**(3), 26–49 (1988)
13. Lamping, J., Rao, R., Pirolli, P.: A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems, CHI 1995, pp. 401–408. ACM Press/Addison-Wesley Publishing Co., New York (1995)
14. Livnat, Y., Agutter, J., Moon, S., Erbacher, R.F., Foresti, S.: A visualization paradigm for network intrusion detection, pp. 92–99 (2005)
15. Livnat, Y., Agutter, J., Moon, S., Foresti, S.: Visual correlation for situational awareness. In: IEEE Symposium on Information Visualization, INFOVIS 2005, pp. 95–102. IEEE (2005)
16. Melancon, G., Herman, I.: Circular drawings of rooted trees. In: Reports of the Centre for Mathematics and Computer Sciences (1998)
17. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 109–118. ACM (2004)
18. Noel, S., Jajodia, S.: Understanding complex network attack graphs through clustered adjacency matrices. In: ACSAC, pp. 160–169 (2005)

19. Xinming, O., Govindavajhala, S., Appel, A.W.: MulVal: a logic-based network security analyzer. In: 14th USENIX Security Symposium, pp. 1–16 (2005)
20. Prautzsch, H., Boehm, W., Paluszny, M.: Bézier and B-Spline Techniques. Springer, New York (2002)
21. Schiffman, M.: The common vulnerability scoring system (CVSS), November 2005
22. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: 2002 Proceedings of the IEEE Symposium on Security and Privacy, pp. 273–284. IEEE (2002)
23. Shneiderman, B., Wattenberg, M.: Ordered treemap layouts. In: 2001 IEEE Symposium on Information Visualization, INFOVIS 2001, pp. 73–78 (2001)
24. Williams, L., Lippmann, R., Ingols, K.: GARNET: a graphical attack graph and reachability network evaluation tool. In: Visualization for Computer Security, pp. 44–59 (2008)
25. Williams, L., Lippmann, R., Ingols, K.: An interactive attack graph cascade and reachability display. In: VizSEC 2007, pp. 221–236 (2008)