Reducing Intrusion Alert Trees to Aid Visualization

Eric Ficke[1(✉)], Raymond M. Bateman[2], and Shouhuai Xu[3]

[1] The University of Texas at San Antonio, San Antonio, USA
eric.ficke@utsa.edu
[2] U.S. Army Research Laboratory South - Cyber, San Antonio, USA
[3] University of Colorado Colorado Springs, Colorado Springs, USA

Abstract. Cyber defense tools, such as intrusion detection systems, often produce huge amounts of alerts which must be parsed for defensive purposes, particularly cyber triage. In this paper, we utilize the notion of *alert trees* to represent the collection of routes that may have been used by a cyber attacker to compromise a set of computers. Although alert trees can be visualized to aid analysis, their usefulness in practice is often discounted by the fact that they can become unmanageable in size. This makes it difficult for cyber defenders to identify patterns or pinpoint network hotspots in order to prioritize defensive maneuvers, raising the need to reduce strain on defenders by minimizing the presence of non-critical information. To address this problem, we propose several methods, as well as a novel data structure, for modifying alert trees in order to reduce visual strain on defenders. We evaluate our methods using a real-world dataset, which demonstrates that our methods are effective at reducing redundancy while limiting collateral information loss.

Keywords: Alert tree · Cyber triage · Visualization · Hypotree · Information loss · Intrusion detection · Network security

1 Introduction

Real-world cyber defense tools often produce a huge number of alerts on a daily basis. It is an important problem to leverage these alerts for defense purposes because they are often the first opportunity for the defender to detect attacks. A common approach for this problem is to use graph-based visualization. However, large graphs can be difficult to analyze manually. It is important to enable this process because human defenders may be able to detect attacks or make sense of alerts that automated tools cannot. Nevertheless, the practice of maintaining "human in the loop" decision-making is often overlooked.

Alert trees have been proposed as an alternative to arbitrary network graphs (or multigraphs). Intuitively, alert trees offer several advantages over graphs: The first advantage is their *planarity*, under which no edges overlap [14]. This is important because planarity makes it easier to visually distinguish edges.

X. Yuan et al. (Eds.): NSS 2022, LNCS 13787, pp. 140–154, 2022.
https://doi.org/10.1007/978-3-031-23020-2_8

The second advantage is that alert trees show the *temporal relationships* among alerts. Graphs prioritize spatial relationships, and can only model temporal relationships by either adding dynamic animations, which take time to observe, or annotating edges with timestamps, which require the effort of granular inspection and interpretation.

Alert trees themselves exhibit some limitations, such as: (i) trees can be prohibitively large; and (ii) trees may present redundant information, which can confuse defenders. Thus it is not trivial to use alert trees to represent the temporal relationships between alerts. This motivates the present study.

1.1 Our Contributions

In order to aid the visualization of alert trees, we hereby contribute a data structure, three algorithms, two metrics and a case study. First, we introduce the concept of *hypotree*, which is useful in the set of reductions that follow by identifying repeated attack patterns across particular links. Second, we propose several novel algorithms for reducing the size of alert trees. One algorithm for *merging sibling leaves*: this eliminates redundancy while preserving significant threats. One algorithm for *merging sibling branches*: this eliminates redundancy while preserving the underlying structure. One algorithm for *truncating hypotrees*, the aforementioned novel data structure: this reduces redundancy by grouping subsequent co-located alerts. Third, we propose methods and metrics for evaluating the usefulness of the above algorithms. Specifically, we consider the effects of the algorithms on *tree size* and *information retention*. These act as trade-offs, representing the sensitive conflict between not enough and too much information. This trade-off poses as a new challenge because information loss has not been studied in the present context, despite research in other contexts (e.g., data anonymization [18] and data perturbation [19]). Fourth, we demonstrate the usefulness of the proposed approach by applying these three algorithms to a well-known dataset. We measure the reduction in visual strain (i.e., tree size) and compare it to the trade-off in lost information.

1.2 Related Work

Since our study is centered at visualizing alerts to help defenders, we divide the relevant prior studies into the following three categories.

Alert and Attack Trees. There have been studies on leveraging alerts to help defenders, such as: correlating alerts to construct attack scenarios or enable collaborative defense [7,20,28], leveraging alerts to learn attack strategies [21], and alert fusion and reasoning [12]. These approaches are useful for modeling various attack patterns. However, these patterns are often not reliable enough to incorporate into a fully automated system, which could damage systems if deployed too aggressively. In light of this, it remains important for alert-based systems to present models that are intuitive to human defenders.

Alert trees [9] are conceptually related to attack trees [1,5,22]. Attack trees are often used to describe the preconditions that allow attackers to achieve their

goals. Because of this, attack trees are often used to guide the hardening of a network, and constitute preventive measures. By contrast, alert trees are meant to make sense of the alerts produced by cyber defense tools as attacks enter the network. This means that alert trees are more appropriate than attack trees for the sake of cyber triage, which measures the scope of various attacks against a network. Alert trees are detective measures. Because the semantics of alert trees differ from attack trees and arbitrary network graphs, the patterns exhibited by alert trees are likely unique. This motivates us to tailor our approach to the specific case of alert trees. To our knowledge, no other works have targeted the problem of reducing redundancy for alert trees in particular.

Alert Aggregation. While the notion of an alert tree does not necessarily demand alert reduction in general, it may be useful during alert tree construction. In this regard, alert reduction is related to the notion of alert aggregation (see, e.g., [17,23,24]), which aims to reduce alert cardinality in order to improve efficiency. The present work contrasts this by managing redundancies after the trees have been constructed, rather than before or during construction. This difference means that any potential data loss is delayed until further down the processing line and should be easier to recover if necessary.

Graph Visualization. Network visualization has been used to present data to defenders for the purposes of cyber triage [16,26]. These visualizations are primarily targeted at identifying individual attacks or aggregating similar attacks, rather than tracking consecutive attacks in a spatiotemporal context. Other works have used graphs in which nodes represent computers, while arcs represent security events, such as attacks or remote access [11,13]. These works focus on detecting anomalies, rather than tracking attacks deterministically and over time. As mentioned above, planarity is guaranteed in alert trees but not alert graphs. Metrics used to rank and color graphs vary, as alert trees contain multiple types of data such as the type and number of attacks observed, vertex connectedness, and number of paths [1,15]. Some libraries used for graph visualization include Tulip [2], Graphviz [8], and Pajek [3].

1.3 Paper Outline

The rest of the paper is organized as follows. Section 2 introduces the research problem and defines important terms. Section 3 details the methods used. Section 4 presents a case study. Section 5 discusses strengths and weaknesses of the work. Section 6 concludes the present paper with future research directions.

2 Problem Formalization

This section introduces the concepts and terms used throughout the paper and describes the context for their use. It also discusses the research questions we hope to answer.

2.1 Setting and Terminology

We investigate the problem of intuitive and efficient cyber triage. We use the context of an enterprise network, which consists of computers, networking devices, and security devices, is managed by a cyber *defender*, and is targeted by a cyber *attacker* who resides inside or outside the enterprise network. Once the attacker establishes a foothold in the network (through exploits, social engineering attacks, or other means), they conduct lateral movement to compromise additional computers. These attacks leave footprints that can be detected by security devices in order to form *alert paths*. The first computer in an alert path is known as the *origin* and all other computers are considered *victims*. The final computer may also be called the path's *target*.

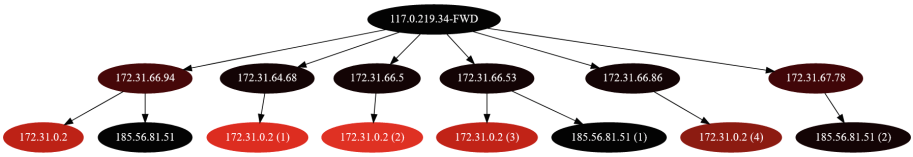


Fig. 1. A forward alert tree, where vertices represent computers (labeled by IP address), arcs represent sets of attacks between computers (according to alerts from security devices), and node colors indicate the threat score of the incoming set of attacks (i.e., the arc from the parent), with red being the highest score. A given computer may appear multiple times, indicating that multiple alert paths exist from the root to that computer. (Color figure online)

When multiple paths branch out from a single origin, these can be formulated into an *alert tree*. Computers in an alert tree are represented by vertices, and arcs between them denote sets of alerts. Alert trees may be forward-looking or backward-looking, such that the root of the tree belongs to all of the tree's paths as either the origin or target, respectively.

The concept of alert trees is important because they serve a critical role in facilitating incident response. Specifically, alert trees help defenders intuitively understand the scope of an attack in terms of the breadth of network impact and focal points thereof.

The visualization of alert trees has presented some significant limitations. Firstly, alert trees have been shown to be particularly large, with some cases resulting in over 5000 nodes, in under a week of attacks [9]. This size of tree is prohibitive to analyze as a whole, but simply removing parts may introduce errors. Thus, the focus of this paper is *reducing visual strain while minimizing information loss*.

2.2 Intuitive Problem Statement

The above discussion naturally leads to several intuitive needs regarding alert tree visualization. Specifically: (i) trees must be reasonably sized for visualization and viewing by defenders, (ii) trees must accommodate or preserve valuable information, and (iii) relevant information on trees must stand out. These problems highlight some of the limitations in the related literature, which offers visualization techniques but does not analyze them for robustness. This inspires us to design and implement the methods here proposed. In what follows we first introduce the concepts used in this paper using formal definitions.

2.3 Data Structures

The core of this work is the reduction operations on alert paths and trees. These concepts are used throughout the paper.

Alert Path. Intuitively, an *alert path* describes a series of attacks traversing one or more network connections, which may have been used by an attacker to conduct a multi-step attack.

Definition 1 (Alert Path). *Given a graph $G = (V, E)$, define an alert path $p = (nodes, edges)$; where $p.nodes = (v_1, v_2, \ldots, v_\ell) \subseteq V$, such that $\forall v_i, v_j \in p.nodes$, $v = v \rightarrow i = j$; and $p.edges = ((v_1, v_2), (v_2, v_3), \ldots, (v_{\ell-1}, v_\ell))$, such that $e \in p.edges \rightarrow e \in E$.*

Alert Tree. An alert tree represents a set of alert paths with a common origin or target and is composed of nodes with corresponding parent/child relationships.

Each node has a name and a color that represents some metric used to show the importance of a node. For this work, we will use the threat score (TS) metric as defined in [9], although the model is metric agnostic. Threat score is used to describe the severity of attacks against a given target. For alert trees, we isolate threat score with respect to a given attacker as well (the node's parent, as described below). It is sufficient to note that node colors range from red to black (i.e., in hexadecimal notation: 0xFF0000 to 0x000000), where red indicates a higher value of the relevant metric, denoting a higher importance. This is demonstrated in Fig. 1, and will be elaborated further in Sect. 3. These nodes are used to construct an alert tree based on the following definition.

Definition 2 (Alert Tree). *An alert tree t is an arborescence (i.e., a rooted directed acyclic graph (DAG) where each node is accessible from the root by a unique sequence of ancestors), rooted at a particular node $t.root$, and for which each node n is annotated by name (denoted $n.name$) and color (denoted $n.color$). Nodes may not share names with any of their siblings or ancestors.*

Alert trees come in two logical forms: forward and backward. For any node n_f in a forward tree, an arc $(n_f.parent, n_f)$ indicates an attack from $n_f.parent$ to n_f. Conversely, for any node n_b in a backward alert tree, an edge $(n_b.parent, n_b)$

indicates an attack from n_b to $n_b.parent$. These are formulated respectively to show the scope of victims that a particular attacker may have targeted, and the scope of attackers that may have targeted a particular victim.

It is also worth noting that alert paths can be reconstructed from an alert tree by extracting a node's ancestors. Note that in the case of backward alert trees, the ancestors must be reversed to retrieve the proper alert path.

Hypotree. Intuitively, a *hypotree* is a tree which resembles a portion of another tree (i.e., its *hypertree*), where the two trees have identical roots. This relationship is distinct from the concept of a subtree, which constitutes a branch of its supertree. By contrast, a hypotree may be missing individual nodes or branches relative to its hypertree.

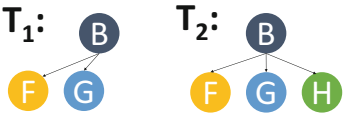


Fig. 2. Example hypotree, where $T_1 \lhd T_2$. Color coding shows analogous nodes in the hypertree. (Color figure online)

Remark 1 ("Hypotree" Usage). The term *hypotree* has been used to refer to an altered subtree structure in [10], but is otherwise absent from the literature. Our usage is not inconsistent with this one. However, this usage may seem to imply that its inverse is a *hypertree*, which has been used to denote an unrelated concept [4,25]. For our purposes, it is sufficient to exclusively use the one-way relationship of hypotree.

Based on the naming restrictions given in the definition of *alert tree*, we can see that for any given alert tree, each node has a single ancestry which is unique in the tree. With this in mind, we define hypotree in Definition 3.

Definition 3 (Hypotree). *A tree T_{hypo} is a hypotree relative to a tree T_{hyper} if $\forall n \in T_{hypo}, \exists n' \in T_{hyper} : n'.ancestors = n.ancestors$*

Denote "T_{hypo} is a hypotree of T_{hyper}" as $T_{hypo} \trianglelefteq T_{hyper}$. Similarly, we derive hypertree ($\trianglerighteq$); proper hypertee ($\rhd$), a hypertree that is not also a hypotree; and proper hypotree ($\lhd$), a hypotree that is not also a hypertree. Recall that "hypertree" is an existing concept in other contexts, so "hypotree" is preferred where possible. Where necessary, the symbol ($\trianglerighteq$) can be used to avoid confusion, as this is specific to the current usage.

An example hypotree is given in Fig. 2. If these were each members of the same alert tree, it would indicate that the T_1 attacks occurred after the T_2 attacks, since consecutive links in an alert path follow a happens-before relationship. In other words, the attacks (B, F) and (B, G) must have come after

the attack (B, H). This means that if the two trees were produced by different attackers and only node H was compromised, we can conclude it was done by the attacker that produced T_2.

2.4 Formalizing Intuitive Problems as Research Questions

Equipped with the preceding formalisms, we can now translate the intuitive problems into rigorous Research Questions (RQs) as follows. Computers may appear multiple times in an alert tree, and thus represent redundant data. Removing this redundant data can improve usability of alert trees. This motivates RQ1 and RQ2.

- **RQ1:** How much can we reduce alert tree size by merging similar nodes?
- **RQ2:** How much can we reduce alert tree size by removing duplicate nodes?

While modifying an alert graph, the removal and merging of nodes can reduce the amount of information available to the defender. Specifically, when merging nodes, we want to maximize the ratio of size reduction to information loss. Similarly, when removing duplicate nodes, we want to preserve to location information of the nodes that were removed. This motivates RQ3.

- **RQ3:** How can we preserve information lost in the solutions to RQ1 and RQ2?

Salient information can represent a wide variety data, such as threat score, asset value, etc. One way to represent such data is to color-code the relevant graph elements. This motivates RQ4.

- **RQ4:** How can we highlight salient information in an alert graph without increasing visual strain on the user?

3 Methods

In this section, we propose several methods for reducing an alert trees. Specifically, we propose merging sibling leaves, merging sibling branches, and truncating hypotrees, as highlighted in Fig. 3.

Each of the three base functions can be used on its own to reduce a given alert tree. These represent reduction schedules one, three and five, respectively. However, because the nodes they merge or remove may overlap, it is unsafe to apply more than one reduction at a time. The only exception is the *merge sibling leaves* reduction, which may be applied after (but not before) either of the other two base reductions because it does not create conflicts with them. This forms reduction schedules two and four.

The model requires alert trees as inputs, as defined in Sect. 2. Once the trees are imported, it annotates them to facilitate the reduction algorithms and sends them to the appropriate functions. The remainder of this section describes the base reductions.

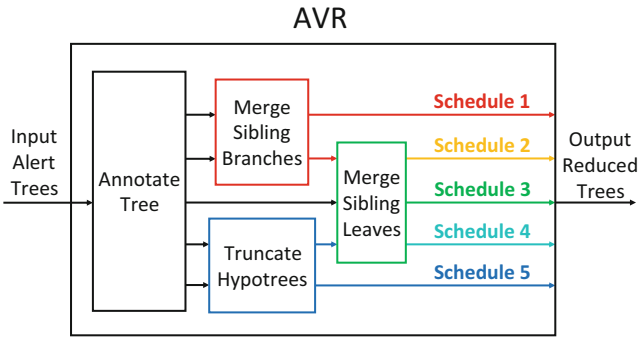
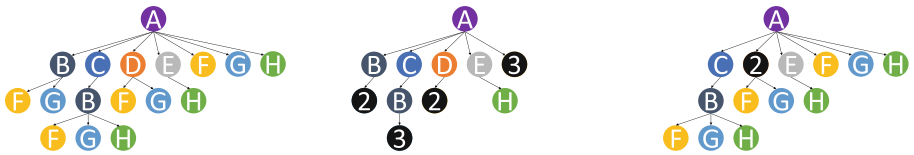
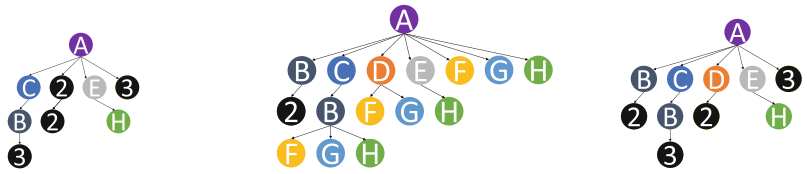


Fig. 3. Reductions overview. The colored boxes represent reductions, while colored arrows represent reduction schedules. Reduction schedules 1,3, and 5 utilize only a single reduction, while 2 and 4 apply two reductions in sequence. (Color figure online)



(a) Base Tree (b) Merged Leaves (ML) (c) Merged Branches (MB)

Fig. 4. Example tree with reduction schedules 3 and 1 applied. Colors show unique nodes. Black denotes merges along with number of nodes merged. (Color figure online)



(a) MB-ML (b) Truncated Hypotrees (TH) (c) TH-ML

Fig. 5. Example reduction schedules 2, 5 and 4. Colors shows unique nodes. Black denotes merges or truncations along with the number of nodes reduced. (Color figure online)

3.1 Merging Sibling Leaves

Intuitively, high volume and low yield information can be reduced by merging sibling leaves, preserving information about the severity of attacks against the

merged leaves according to the TS of the merged leaves. In attack trees, this is done by adopting the color of the node with the highest TS among those merged.

The approach is as follows. We use a breadth-first traversal to iterate over the tree, parsing the list of each internal node's children. For each set of children, we check each node to determine if it is a leaf node and document the highest TS within the set of those leaves. We then replace the leaves with a single node, showing the number of leaves merged and color coded according to the TS selected above. Figure 4b shows the result of merging leaves on the example tree from Fig. 4a.

3.2 Merging Sibling Branches

We can reduce the amount of duplicate information presented to the viewer by merging identical branches. This allows the viewer to more quickly identify common patterns within the tree.

The approach is as follows. In order to find identical branches, we conduct a breadth-first traversal of the tree, recursively comparing the branches of each set of siblings. When we identify two siblings with identical subtrees, we combine the siblings into a single node, preserving the shared form of the subtree. We label the new node with the number of siblings that were merged, applying the same color from the sibling with the highest ETS. Node data for the merge is archived in case it needs to be retrieved later.

Branches are compared using a hash function, which produces a tuple $H(root) = (root, (H(node) : node \in root.children))$, which incidentally produces $H(leaf) = (leaf, ())$ for leaf nodes. Figures 4c and 5a show examples of the *merge branches* reduction.

3.3 Truncating Hypotrees

Because all nodes in hypotrees are duplicated in their respective hypertrees, they are redundant. For this reason, we choose to truncate hypotrees in order to reduce visual strain on the viewer. This preserves the most amount of information since all edges are preserved (in the corresponding hypertree), even if their location and number are lost.

The basic idea of the algorithm is described as follows. We parse the tree using a breadth-first traversal, marking all nodes that share the same address. We then compare the hypotrees of each set of identical nodes, preserving trees which have no proper hypertrees and truncating the rest. In the case of two equivalent hypotrees (i.e., $A \trianglelefteq B \wedge B \trianglelefteq A$), we preserve only the one appearing first in the traversal. Truncated trees contain annotations to refer viewers to the corresponding hypertree. Archives save information about removed hypotrees so they can be reconstructed if needed. The method for truncating hypotrees is given in Algorithm 1. Figures 4b and 4c show an example usage of the algorithm.

Algorithm 1. Truncate Hypotrees

Input: *Root*
Output: *Root*, *Archives*

$Candidates \leftarrow \emptyset$
$Unique\_Names \leftarrow \emptyset$
for each $Node \in Root.descendants$ **do**
  if $\exists Name\_List \in Unique\_Names : Name\_List_1 = Node.name$ **then**
    $Name\_List \leftarrow Name\_List \cup (Node)$
    $Candidates \leftarrow Candidates \cup \{Node.name\}$ ▷ Nodes sharing a name become candidates
  else
    $Unique\_Names \leftarrow Unique\_Names \cup \{(Node.name, Node)\}$
    ▷ Here elements 2 onward are nodes with the same name
  end if
end for
$Trunks \leftarrow (\emptyset^{|Unique\_Nodes|})$
$Colors \leftarrow (()^{|Unique\_Nodes|})$
for each $n \in [1, 2, \ldots, |Unique\_Names|]$ **do**
  if $Unique\_Names_n \in Candidates$ **then**
    for each $i, j \in [2, 3, \ldots, n), i < j]$ **do** ▷ Compare pairs of candidates
      if $Unique\_Names_{n,i} \lhd Unique\_Names_{n,j}$ **then**
        if $|Unique\_Names_{n,i}.descendants| > 1$ **then**
          $Trunks_n \leftarrow Trunks_n \cup (Unique\_Names_{n,i})$ ▷ Mark i for truncation
          $Colors_n \leftarrow Colors_n \cup (\mathsf{max}_{d \in Unique\_Names_{n,i}.descendants}(d.color))$
        end if
      else
        if $|Unique\_Names_{n,j}.descendants| > 1$ **then**
          $Trunks_n \leftarrow Trunks_n \cup (Unique\_Names_{n,j})$ ▷ Mark j for truncation
          $Colors_n \leftarrow Colors_n \cup (\mathsf{max}_{d \in Unique\_Names_{n,j}.descendants}(d.color))$
        end if
      end if
    end for
  end if
end for
$Archives \leftarrow \emptyset$
for $i \in [1, \ldots, |Trunks|]$ **do**
  for $j \in [1, \ldots, |Trunks_i|$ **do** ▷ Truncate and archive
    $trunk\_archive \leftarrow \mathsf{copy}(Trunks_{i,j}.parent)$
    $trunk\_archive.parent \leftarrow \emptyset$
    $new\_trunk \leftarrow \mathsf{copy}(Trunks_{i,j})$
    $new\_trunk.color \leftarrow Colors_{i,j}$
    $Trunk_{i,j}.parent \leftarrow trunk\_archive$
    $Archives \leftarrow Archives \cup \{trunk\_archive\}$
  end for
end for
return *Root*, *Archives*

4 Case Study

For the case study, we used CSE-CIC-IDS2018 [27], a well-known dataset collected from a testbed with both injected and wild attacks. From the network traffic, Snort [6] produced 3.3M alerts. These were assembled into trees using APIN [9]. Nodes were ranked according to threat score, calculated as the geometric mean of the volume and diversity of alerts incident to the node. Trees were ranked according to the threat score of their root node.

From the resulting trees, we selected 15 for AVR to reduce: 5 each from the top ranked, bottom ranked, and randomly selected trees. Statistics for the selected trees are as follows. The top 5 set had on average 9.8 vertices, 7.6 unique vertices and 8.8 unique arcs. Next, the bottom 5 set had on average 15 vertices, 10.6 unique vertices and 14 unique arcs. Finally, the random 5 set had on average 1999 vertices, 234.4 unique vertices and 825.6 unique arcs.

4.1 Evaluation Metrics

In order to evaluate the effectiveness of our methods, we utilize a total of 4 metrics, including three atomic metrics and one aggregate metric. The atomic metrics are visual strain reduction (VSR), node retention (NR) and threat score retention (TSR). The latter two are derived from the notion of *information loss*, as its additive inverse (i.e., $1 - loss$). These three metrics are also combined to create a *reduction index*.

For the first metric, we measure VSR as the number of nodes in the reduced tree relative to the full tree. VSR has a range of [0,1], where 100% is ideal.

For the following two metrics, we measure *information retention* as the number of unique nodes or threat score values from the full tree that remain after the reduction. For unique nodes, recall that a given computer may appear multiple times in an alert tree because there may be multiple paths that an attacker may have taken to reach it. If the unique node remains in the tree after the reduction, it is considered retained. For threat score values, recall that threat score describes the severity of a set of attacks between two nodes. Thus, if the corresponding color (for the node representing the target of those attacks) remains in the tree, the corresponding TS is considered retained. Both NR and TSR have a range of [0,1], where 100% is ideal.

Note that NR is not necessarily $1 - \text{VSR}$, since some reductions add supplemental nodes after pruning. These new nodes increase visual strain but not node retention, since they do not belong to the full tree. However, they may increase TSR, since some of the supplemental nodes inherit color codes (i.e., threat score) from the node(s) they replaced.

To ensure a balance between size reduction and information retention, we combine the three metrics into a *reduction index* using their harmonic mean.

4.2 Results

Results of the experiments are given in Table 1. Of the basic reductions, the *truncate hypotrees* performed the worst in most cases, with its RI trailing by

margins of 0.373 and 0.354 for the random 5 and bottom 5 categories, respectively. In the top 5 category, however, it outperformed both other algorithms by at least 0.131.

Table 1. Results evaluated on alert trees sampled according to threat score. VSR is average visual strain reduction, NR is average node retention, TSR is average threat score retention, and RI is reduction index as the harmonic mean of VSR, NR and TSR.

Algorithm	VSR	NR	TSR	RI
1. Merge branches (top 5)	0	1	1	0
1. Merge branches (rand 5)	.448	.533	.254	.373
1. Merge branches (bot 5)	.567	.493	.360	**.457**
2. MB-ML (top 5)	.977	.008	.010	.013
2. MB-ML (rand 5)	.561	.357	.929	**.530**
2. MB-ML (bot 5)	.761	.205	.120	.206
3. Merge leaves (top 5)	.977	.008	.010	.013
3. Merge leaves (rand 5)	.118	.824	.799	.274
3. Merge leaves (bot 5)	.209	.255	.270	.242
4. TH-ML (top 5)	.981	.008	.010	.013
4. TH-ML (rand 5)	.118	.824	.201	.204
4. TH-ML (bot 5)	.239	.743	.713	.433
5. Trunc hypotrees (top 5)	.053	1	1	**.144**
5. Trunc hypotrees (rand 5)	0	1	1	0
5. Trunc hypotrees (bot 5)	.037	1	.983	.103

Overall, reduction schedule 2 performed the best in the random 5 category, and schedule 4 performed the best in the bottom 5 category. This suggests that merging branches then leaves is the best general-purpose strategy, while truncating hypotrees then merging leaves then is the best strategy for reducing relatively small trees.

With respect to any one particular metric, results varied across reduction schedules and sample sets. This means it may be difficult to predict which schedule one should use when trying to optimize for any particular metric.

4.3 Answering Research Questions

Answering RQ 1: How much can one reduce alert tree size by merging similar nodes? By merging leaves, tree size can be reduced by as much 98%, and by merging branches tree size can be reduced by as much as 57%.

Answering RQ 2: How much can one reduce alert tree size by removing duplicate nodes? By truncating hypotrees, tree size can be reduced by as much as 5.3%.

Answering RQ 3: How can one preserve the information lost in the solutions to RQ1 and RQ2? The best way to preserve information is to truncate hypotrees, which contain almost exclusively redundant information. Otherwise, results from the other algorithms have variable results depending on the sample used.

Answering RQ 4: How can one highlight salient information in an alert graph without increasing visual strain on the user? Color-coding salient information allows the tree to highlight important data such as network hotspots and threat activity. Color can be used for both nodes and edges, so NR and TSR are the metrics to look at when one needs information salience.

The novel reductions had a broad range of performance, with each one having a different strength. Since user needs will vary, it will be important to consider these differences when choosing how to handle alert trees. Meanwhile, these results are only preliminary and warrant further study.

5 Discussion

Limitations of the Methodology. The methods used in this study have the following limitations. First, the reductions for merging branches and truncating hypotrees have overlapping domains under composition. This means running them in sequence may give different results depending on the order used. This results in only five valid reduction schedules.

Additionally, alert paths do not necessarily give a precise account of an attacker's activity. This is for the following reasons: (i) attacks may fail, producing alerts that do not indicate compromise; (ii) attacker addresses may be spoofed or reflected, such that the source of the connection is not visible to network monitors; (iii) security devices may have false positives or negatives; and (iv) some attacks may use client-side exploits, resulting in arcs that are inverted (i.e., the compromised computer may be the source of an attack rather than its destination). These phenomena can induce errors in the experimental results.

Limitations of the Case Study. The dataset in the case study utilizes threat score to rank edges and paths. This metric has not been robustly studied and may not produce the best scores relative to a particular attack. However, the methods proposed in the present study need not use threat score, but could easily be adapted to rank nodes according to monetary value, vulnerability score, or other related risk metrics.

6 Conclusion

This work introduced several methods for reducing the size of alert trees while retaining as much information as possible. The three core functions can be used

independently or combined in a total of five reduction schedules. One of the reductions involves the use of a novel data structure, the hypotree. These reductions were applied to alert trees from a research testbed dataset, and were evaluated for information retention and visual strain reduction. Results show that the reductions have varied performance relative to each other for different types of trees. This finding warrants more research into how the application of the reductions may be optimized.

Acknowledgments. This work was supported in part by NSF Grants #1736209, #2122631 and #2115134, and by Colorado State Bill 18-086.

References

1. Angelini, M., Prigent, N., Santucci, G.: PERCIVAL: proactive and reactive attack and response assessment for cyber incidents using visual analytics. In: 2015 IEEE Symposium on Visualization for Cyber Security (VizSec), pp. 1–8. IEEE (2015)
2. Auber, D.: Tulip—a huge graph visualization framework. In: Jünger, M., Mutzel, P. (eds.) Graph Drawing Software. Mathematics and Visualization, pp. 105–126. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-642-18638-7_5
3. Batagelj, V., Mrvar, A.: Pajek-program for large network analysis. Connections **21**(2), 47–57 (1998)
4. Brandstädt, A., Chepoi, V.D., Dragan, F.F.: The algorithmic use of hypertree structure and maximum neighbourhood orderings. Discret. Appl. Math. **82**(1–3), 43–77 (1998)
5. Chen, Y., Boehm, B., Sheppard, L.: Value driven security threat modeling based on attack path analysis. In: 2007 40th Annual Hawaii International Conference on System Sciences (HICSS 2007), pp. 280a–280a. IEEE (2007)
6. Cisco: Snort - network intrusion detection & prevention system, March 2018. http://www.snort.org/downloads
7. Cuppens, F., Miège, A.: Alert correlation in a cooperative intrusion detection framework. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, SP 2002, p. 202 (2002)
8. 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). https://doi.org/10.1007/3-540-45848-4_57
9. Ficke, E., Xu, S.: APIN: automatic attack path identification in computer networks. In: IEEE ISI 2020 (2020)
10. Gerbessiotis, A.V.: An architecture independent study of parallel segment trees. J. Discrete Algorithms **4**(1), 1–24 (2006)
11. Goodall, J.R., et al.: Situ: identifying and explaining suspicious behavior in networks. IEEE Trans. Vis. Comput. Graph. **25**(1), 204–214 (2019)
12. Gu, G., Cárdenas, A., Lee, W.: Principled reasoning and practical applications of alert fusion in intrusion detection systems. In: Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS 2008), pp. 136–147 (2008)

13. Harshaw, C.R., Bridges, R.A., Iannacone, M.D., Reed, J.W., Goodall, J.R.: GraphPrints: towards a graph analytic method for network anomaly detection. In: Proceedings of the 11th Annual Cyber and Information Security Research Conference, CISRC 2016, pp. 15:1–15:4. ACM, New York (2016). https://doi.org/10.1145/2897795.2897806
14. Herman, I., Melançon, G., Marshall, M.S.: Graph visualization and navigation in information visualization: a survey. IEEE Trans. Visual Comput. Graphics **6**(1), 24–43 (2000)
15. Kerzner, E., et al.: Graffinity: visualizing connectivity in large graphs. In: Computer Graphics Forum, vol. 36, pp. 251–260. Wiley Online Library (2017)
16. Lohfink, A.P., Anton, S.D.D., Schotten, H.D., Leitte, H., Garth, C.: Security in process: visually supported triage analysis in industrial process data. IEEE Trans. Visual Comput. Graphics **26**(4), 1638–1649 (2020)
17. Nadeem, A., Verwer, S., Yang, S.J.: SAGE: intrusion alert-driven attack graph extractor. In: 2021 IEEE Symposium on Visualization for Cyber Security (VizSec), pp. 36–41. IEEE (2021)
18. Nettleton, D.F.: Information loss evaluation based on fuzzy and crisp clustering of graph statistics. In: 2012 IEEE International Conference on Fuzzy Systems, pp. 1–8. IEEE (2012)
19. Nettleton, D.F., Torra, V., Dries, A.: The effect of constraints on information loss and risk for clustering and modification based graph anonymization methods. arXiv preprint arXiv:1401.0458 (2014)
20. Ning, P., Cui, Y., Reeves, D.S.: Constructing attack scenarios through correlation of intrusion alerts. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, pp. 245–254 (2002)
21. Ning, P., Xu, D.: Learning attack strategies from intrusion alerts. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, pp. 200–209 (2003)
22. Ray, I., Poolsapassit, N.: Using attack trees to identify malicious attacks from authorized insiders. In: di Vimercati, S.C., Syverson, P., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 231–246. Springer, Heidelberg (2005). https://doi.org/10.1007/11555827_14
23. Sadoddin, R., Ghorbani, A.: Alert correlation survey: framework and techniques. In: Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services, pp. 1–10 (2006)
24. Salah, S., Maciá-Fernández, G., Díaz-Verdejo, J.E.: A model-based survey of alert correlation techniques. Comput. Netw. **57**(5), 1289–1317 (2013)
25. Schidler, A., Szeider, S.: Computing optimal hypertree decompositions. In: 2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 1–11. SIAM (2020)
26. Sethi, A., Wills, G.: Expert-interviews led analysis of EEVi-a model for effective visualization in cyber-security. In: 2017 IEEE Symposium on Visualization for Cyber Security (VizSec), pp. 1–8. IEEE (2017)
27. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP, pp. 108–116 (2018)
28. Valeur, F., Vigna, G., Kruegel, C., Kemmerer, R.A.: A comprehensive approach to intrusion detection alert correlation. IEEE Trans. Dependable Secur. Comput. **1**(3), 146–169 (2004)