Visualizing Traffic Causality for Analyzing Network Anomalies*

Hao Zhang, Maoyuan Sun, Danfeng (Daphne) Yao, and Chris North
Department of Computer Science
Virginia Tech, Blacksburg, VA
{haozhang, smaoyuan, danfeng, north}@cs.vt.edu

ABSTRACT

Monitoring network traffic and detecting anomalies are essential tasks that are carried out routinely by security analysts. The sheer volume of network requests often makes it difficult to detect attacks and pinpoint their causes. We design and develop a tool to visually represent the causal relations for network requests. The traffic causality information enables one to reason about the legitimacy and normalcy of observed network events. Our tool with a special *visual locality* property supports different levels of visual-based querying and reasoning required for the sensemaking process on complex network data. Leveraging the domain knowledge, security analysts can use our tool to identify abnormal network activities and patterns due to attacks or stealthy malware. We conduct a user study that confirms our tool can enhance the readability and perceptibility of the dependency for host-based network traffic.

Keywords

Anomaly Detection, Network Traffic Analysis, Information Visualization, Usable Security, Visual Locality

1. INTRODUCTION

This paper addresses the issue of visualizing the network traffic causality. We aim to design a visualization tool to facilitate the process of identifying anomalous network traffic. The recently proposed detection method advances the analysis of network traffic by inferring the semantic and logical relations [25]. Its unique advantage is the capability of reasoning the causality or dependency of network data and thus detecting new stealthy malware activities. The analysis provides automatic anomaly detection in the observed network activities through probabilistic reasoning of the causal relations in traffic. By pinpointing abnormal network events that lack of valid triggers, it can detect malware activities on an infected machine (e.g., making command-and-control communications with its controller). The triggers include legitimate user events and benign network packets.

In this work, we design a tool to assist the analysis of host-based network data based on traffic causality. Our visual representation improves the sensemaking process for security and can increase the productivity for analysts.

Many existing network security visualization tools provide graphic user interfaces for Intrusion Detection Systems logs (e.g., Snort) [3, 5, 13, 18]. IDS alerts are organized in a log-type structure, where each alert entry indicates a potential intrusion threat. However, very few existing work provides the visualization of underlying relationship among network events, with one notable exception Portall [7]. Portall visualizes the correlation of host processes and network activities. Our request-level traffic causal relations are much more fine-grained than the process-level correlation in [7]. Thus, new visual representation approaches are needed. Our solution aims to satisfy a unique space efficiency requirement, that is, *how to optimally utilize the screen space for displaying the causal relations of a massive amount of network traffic.*

A straightforward approach for displaying host-based traffic is shown in Figure 1a. This visual representation arranges network requests using a forest layout based on their causal relations; the timeline may be extended horizontally when newer network events are added. Because the forest-based layout is intuitive, it has been used for illustrating relations among network events [14, 25]. However, it does not use the display space efficiently. The length of traffic causality structure grows fast, making it difficult to view related events that are temporally far apart. Statistics shows that more than 90% of the request causal relation falls within a 30-second interval [25]. Although rare, we observe that network requests that occur 15 minutes apart may have causal relations. Therefore, our visualization design takes these unique traffic characteristics into considerations. To this end, we focus on displaying items that have causal relations in visually adjacent space. Causal relations determine their locations on the display. This layout provides two advantages: *i)* enhancing the navigation of traffic causality, and *ii)* improving the identification of anomalous activities.

Our contribution in this work is twofold.

- We develop a visualization tool for security analysts to efficiently display the network traffic dependency. This tool has a *visual locality* feature that can optimize the displaying of structured data. The visual representation is a *radial* layout based on curved timeline display.

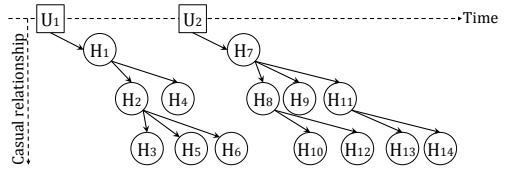
*This work has been supported in part by NSF grant CAREER CNS-0953638, ARO YIP W911NF-14-1-0535, and L-3 communications.

IWSPA'15, March 4, 2015, San Antonio, Texas, USA.

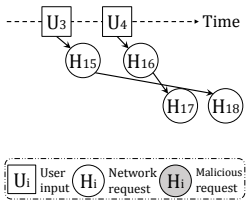
http://dx.doi.org/10.1145/2713579.2713583 .

Compared to the straightforward forest layout, our design maximizes the use of the screen by bending the timeline into a circle, which achieves high visual locality and extensibility.

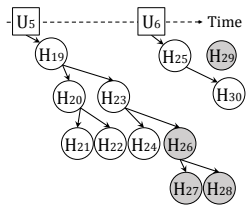
- We conduct a user study to evaluate our system with 10 participants and real-world network traffic. Results of the study show that our tool is well suited for security analysts to perform manual inspection and analysis on network events based on their causal relations.



(a) A straightforward representation of TRG.



(b) Two crossing edges.



(c) Malicious requests.

Figure 1: Schematic drawing of traffic causality for outbound network requests on a computer. Nodes are indexed by their relative occurrence time.

2. TRG AND SECURITY MODEL

DEFINITION 1 (**Triggering Relation Graph** [25]). *TRG is composed of two types of nodes, user input and network request. The edges in TRG refer to the triggering relations that describe the causal relationship between nodes. The dependencies are defined by two types of edges: i) root-trigger dependency defines the relation between a legitimate user's input (e.g., mouse clicks on hyperlinks) and its generated first network request and ii) inter-request dependency is the relation between two network requests where one directly triggers the other.*

Figure 1a shows an example of a TRG, where $U_1 \rightarrow H_1$ is an example of a root-trigger dependency edge and $H_1 \rightarrow H_2$ is an example of an inter-request dependency edge.

Triggering relations can be computed by using rule-based methods [24] or machine learning techniques [25]. TRG is built on the application layer packets and used to find all triggering relations to understand how a user interacts with applications and how applications respond to the user by sending out network requests. Therefore, it reveals the logical structure of the requests, which can be used to detect abnormal network activities originated from the host. In this model, the events having triggering relations to a legitimate cause are benign ones.

However, there is no existing tool to display the triggering relation graph. Straightforward attempts to visualize the TRG (e.g., using conventional straight timeline layout) may not satisfy the high visual locality requirement. In addition, the requests with causality do not necessarily situate close according to the time, as illustrated in Figure 1b (H_{15}-H_{18}). The crossed edges make the TRG messy and hard to analyze.

DEFINITION 2 (**High visual locality**). *Items having logical relations are placed close to each other on display.*

In our context, we define *high visual locality* as our primary goal to optimize our visualization designs. It enables analysts to easily identify related requests. To meet this requirement, our design prioritizes the causality that clusters nodes around their root-triggers and forms trees separately. Within each tree, the nodes are organized by their temporal and other logical information. In our security model, we consider two types of stealthy malicious network traffic.

- Network requests without valid root triggers are referred to as *vagabond* (H_{29} in Figure 1c).
- Network requests sent to malicious hosts with valid referrer information are referred to as *grafted* (H_{26}-H_{28} in Figure 1c).

The vagabond requests are events without legitimate causal relations and likely due to stealthy malware activities. The grafted requests take place when the servers are misconfigured or compromised, and thus are hard to formalize rules to identify. Blocking the malware network activities effectively isolates the malware, such as spyware exfiltrating sensitive information through outbound traffic. Our TRG model is general and needs no *priori* knowledge about a particular malware class.

Analysis Using All Triggering Relations. One of the design choices for visualizing TRG is whether or not to display *all* the discovered triggering relations. A simple visual representation is to only display vagabond requests, i.e., abnormal network events. However, analysts may neglect some attacks by viewing this type of display, because suspicious requests could be hidden from legitimate ones (i.e., the *grafted* requests). For example, a common attack on web servers is to exploit web vulnerabilities (e.g., SQL injection, cross site scripting, format string injection [8]). After the servers are compromised, attackers can modify the websites by injecting malicious codes, then the clients get infected when they visit the websites. These grafted requests are often of the `Javascript` type with long and obfuscated request strings in URLs, because `Javascript` requests have diversified functions to be leveraged by attackers. Domain experts have to reason the legitimacy by integrating with other information (e.g., system logs).

Our TRG model provides a good visual representation for a host-based overview. The *grafted* requests, sent by compromised servers, can be identified by their deeper levels in TRG, late-arriving timestamps, and unusual host domains. Therefore, the analysts need to leverage the inner logic to infer the legitimacy of requests, which makes the displaying of *all* triggering relations more desirable.

3. VISUALIZATION DESIGN

Our design for the visualization tool is based on characteristics of traffic triggering relations. We run a pilot study that contains 12MB network data (10-hour HTTP traffic on a host, 45000+ requests) to investigate the characteristics of traffic dependency. We summarize our findings as follows.

Wide-and-shallow trees. The nodes on the top three levels in TRG account for above 90% of the total amount. There are respectively 68% and 21% of requests on the second and third levels, so the trees are extremely wide on their

top levels. Besides, 99.7% of trees in TRG have less than 6 levels, while there is only 0.3% of trees whose depth is 7 to 9 levels, which illustrates the shallowness of the trees.

Temporally adjacent events. We further check the time difference between any two requests that have a triggering relationship. Statistics shows that about 93% of HTTP requests trigger their dependencies within 3 minutes. If the time window is enlarged to 15 minutes, then 99.8% of HTTP requests and their dependencies are included. Therefore, the HTTP requests with dependents are temporally close to each other, despite some rare cases.

Sparsity of vagabonds. The vagabond requests are classified into two groups. Malicious and misconfigured packets represent 0.3% and 0.5% respectively of the total number of nodes. *Malicious* requests are sent when a user visits a compromised website or the host is infected by malware. *Misconfigured* requests are not sent to malicious hosts but contain some missing fields in the request header.

Automatic update requests. The updates are the legitimate requests sent to upgrade the system or software periodically, without user interaction. In our design, we maintain a list of known programs and their official upgrade domains to reduce the false alarms (e.g., `Java Update Checker` and its update domains `javadl-esd.sun.com` and `javadl.oracle.com`).

Our design aims to visualize the traffic dependency of network requests by meeting the *high visual locality* principle, so users can identify and analyze the anomalies with ease. We utilize a *radial* design for displaying traffic triggering relations. This design has a curved timeline that is centered at the display, and the radiating branches represent network traffic events and their trigger relations. The advantage of this view design is that it maximizes the utilization of the display screen. It is more convenient for users to interact with this view than a conventional straight timeline view. We further provide a condensed view to simplify the display by merging the trivial nodes. In addition, our design uses a *heatmap* to show the distribution of the requests over time and store original logs in each color-coded tile.

3.1 Visual Locality Design for Analysts

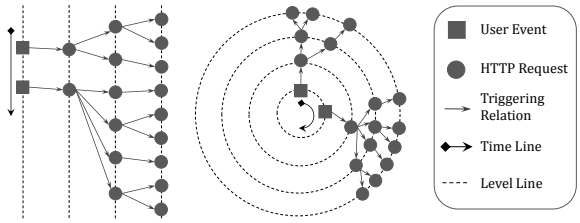
We design a radial layout to display the traffic dependency with *high visual locality*. A straightforward visualization of TRG would be an axial layout in Figure 2a, however it is not suitable for visualizing large-scale traffic dependencies. This layout is not efficient at displaying hierarchy structures, as the tree structure spans unilaterally and leads to much unused space. In addition, the length of the forests grows as the data size increases. Users have to scroll up and down for browsing and searching. In the radial layout shown in Figure 2b, we arrange the nodes in a clockwise manner. In this design, *time line* defines the positive direction and the start point is at 12 o'clock position, which is consistent with an analog clock and intuitive to users.

Our prototype arranges the nodes and allocates the space of the radial layout as follows.

(1) We sort the root-triggers by their timestamps and plot them in the innermost ring. We cluster the nodes under their root-triggers. Clustering nodes of each tree guarantees there is no cross edges in the display.

(2) The rendering space is allocated in proportion to the number of nodes on its *second level*, rather than the timespan of the tree in TRG. The time spans of each tree in TRG may overlap. Requests from different trees may arrive simultaneously. Our layout, based on indexing nodes, eliminates the overlapping issue.

(3) We render the nodes by their levels. The *level lines* are used to align the nodes on the same level. The nodes on the same level are lined up on the concentric arcs. The innermost ring is used to place the user inputs, and is divided into sectors whose angles correspond to the sizes of its dependent nodes.



(a) Axial layout. (b) Radial layout.

Figure 2: Schematic diagrams of two visualization designs for the structured network data.

Our design maximizes the usage of display space. The radial layout presents hierarchies of events in concentric rings. Users using the conventional forest-like layout need to scroll *twice* more than our design. Additionally, by concentrating nodes at the center of the screen, the radial layout enables users to easily manipulate the display.

3.2 Interactive Heatmap for Accessing Original Traffic Logs

Heatmap is a graphical data presentation approach where each value in the matrix is color-coded. Our design is composed of `LogMap`, an instance of Heatmap (see Figure 4). The `LogMap` reveals the density information of the network requests. It provides an overview of the request distribution over the observed period. The `LogMap` accommodates different levels of time-period granularities, e.g., seconds, minutes, hours, etc. Last, security analysts often resort to the original logs for more details. In our design, the `LogMap` supports users to access the original logs by clicking on the colored tiles in the heatmap.

The `LogMap` divides the timeline into fixed windows and organizes network events occurred in each time window into a sub-block. In our design, there are sixty tiles in a sub-block, which represents sixty seconds. The color coding in each tile corresponds to the number of requests. The granularity of events displayed on the `LogMap` can be adjusted according to analysts' needs.

3.3 Condensed View to Distill Information

According to our pilot study, more than 90% of network requests are situated on the top three levels in the TRG. To avoid visual clutter, we provide a condensed view for security analysts. We design a condensing algorithm to merge the nodes that meet *all* the following criteria.

- Legitimate requests that are of the same type;
- Requests that are on the same level in the TRG;
- Requests that are the leaf nodes in the TRG.

The algorithm iterates a list of chronologically sorted requests and outputs a list of condensed nodes. We use an

auxiliary dictionary to store each newly generated node and a list of requests being condensed. In the condensing algorithm, we only merge the benign requests, so as to avoid losing any information of abnormal requests. Our condensing algorithm does not merge the nodes on the different (sub)trees, which guarantees that the dependency structure in a TRG is preserved. Therefore, the condensed views are compatible to the original radial and axial layouts.

We categorize the HTTP requests into six types, which are web, CSS, Javascript, multimedia, data, and others. Around 50% of browser-generated HTTP requests are used to fetch the multimedia objects (e.g., image or streaming data) in our pilot study. Unlike Javascript objects, these requests to obtain *static* files do not trigger further HTTP traffic.

To evaluate the effectiveness of the condensing algorithm for reducing the redundancy and emphasizing the anomalies, we test our tool on the pilot study dataset. Shown in Table 1, we compare the number of nodes in both original and condensed views. *Compression ratio* is defined to qualify the effectiveness of the algorithm. The root-triggers are on the first level, and thus cannot be compressed. There are 68% of total requests situated on the second level, and 87.5% of them are merged, which significantly saves space. Overall, the total compression ratio is 82.2%. Multimedia requests mostly serve as leaf nodes and can be compressed as much as 91%. Compared with the original view, our condensed view significantly reduces the redundancy of displaying leaf nodes. Therefore, it helps users identify abnormal nodes due to its visually salient.

Level in TRG	# of nodes in original view (n)	# of nodes in condensed view (c)	Compression ratio ($1 - \frac{c}{n}$)
1	1158	1158	0.0%
2	31242	3913	87.5%
3	9725	2190	77.5%
4	2753	644	76.6%
5	835	225	73.1%
6	201	49	75.6%
7 - 9	74	24	67.6%
Total	45988	8203	82.2%

Table 1: The number of HTTP requests on each level in the Triggering Relation Graph for original and condensed views.

4. PROTOTYPE IMPLEMENTATION

We build ReView, a visualization tool for viewing and analyzing the triggering relations for network requests. It is designed based on the three-tier architecture and implemented as a web-based tool. The workflow is illustrated in Figure 3.

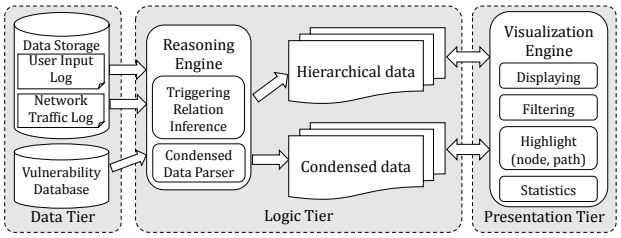


Figure 3: The workflow of ReView.

ReView takes the user events (e.g., clicking on a hyperlink of a webpage) and the outbound HTTP requests as inputs. The network requests are recorded by leveraging the `libpcap` library. The features of an HTTP request include its timestamp, process ID, source and destination IP address, and request semantic information (e.g., HTTP host domain and referrer). In the `data tier`, we add a customized vulnerability database, which is composed of several known blacklists and feeds [15, 16, 23], to facilitate the process of identifying the suspicious requests. However, solely relying on the blacklists is not sufficient because of the ever-changing malware variants and malicious domains.

In the `reasoning engine`, our tool infers the triggering relations by leveraging a machine learning-based solution proposed in [25]. It discoveries vagabond request that are not generated by any user actions. ReView screens out the benign update traffic using a whitelist. It also filters the traffic according to our customized vulnerability database. Other advanced solutions to rank domains and detect malicious URLs (e.g, [21]) can be also integrated into our tool. In addition, we leverage the condensing algorithm to preprocess the raw traffic data in the `logic tier`.

The rendering in ReView uses `D3` [6], which is a Javascript library for data visualization. The layout of ReView is shown in Figure 4. Our design contains two major components, *main display* and *heatmap* panels.

On the main display panel, we show the optimized visualizations of hierarchical structure using the radial layout. The user inputs are placed in the innermost ring. The HTTP requests are situated on the second and later levels. On the right hand side, a draggable control panel is composed of five tabs that are used to manipulate the layout options, query the source data, highlight the nodes, and show the statistics. Our tool supports path highlighting for exploratory analysis. The path from a selected node to its root-trigger can be highlighted, which helps security analysts identify the *logic chain* of the nodes, understand why the request is triggered and find the common ancestors for multiple nodes. Our tool aims at presenting the causal relations, meanwhile minimizing the display of the detailed information for each request. In ReView, there are three ways to display the HTTP request information: *i)* using the `Popup` when mouse hovers over a node; *ii)* reading the information at `InfoTab` after clicking a node; and *iii)* loading the complete information in a separate `LogWindow`. These options can reveal the details of different levels, as requested by users.

On the heatmap panel, the `LogMap` reveals the traffic patterns, which is complementary to the main panel. Each tile in `LogMap` represents one second and is colored on a green scale based on the number of requests in this second. The tile with a red frame indicates that at least one vagabond HTTP request is observed during its time window. A separate window that displays the original logs is shown when a colored tile is clicked.

5. USER STUDY

We carried out a user study with ten participants, and all of them have at least four years of experience working with computers. Their specialties include system/network security, high performance computing and human computer interaction. As our tool is to visualize the network requests for security purposes, it is not built for the average users. We target users with computer science knowledge and people who are curious about, or care about, the computer security. In this user study, we investigate: *i)* the user's preference, and the trade-off between a neat view with condensed data

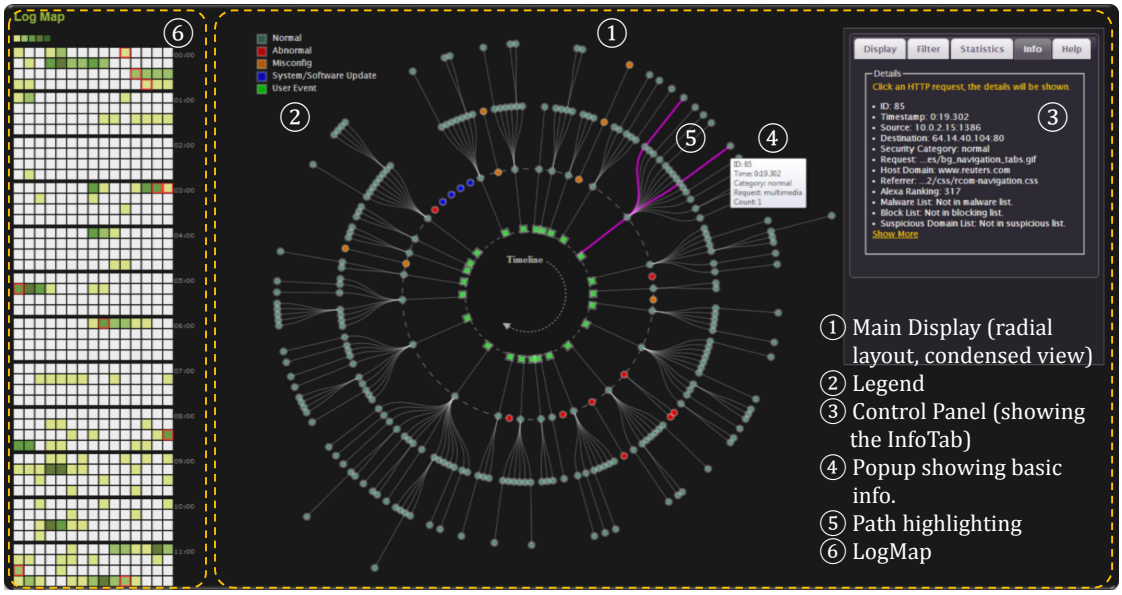


Figure 4: A screen layout of ReView.

and a complete view with all triggering relations; *ii)* how our tool can direct users to analyze the suspicious requests.

We conducted the study in a computer science laboratory in a university. A 15-minute tutorial was given to the participants to introduce the functionalities of ReView. We directed the tutorial using a 7-page slide presentation, so that every participant got the equivalent level of details. In the study, participants were asked to finish 10 questions that include the tasks of analyzing logs routinely done by analysts and the user preference of different visualization options.

The data used in the user study was obtained from a graduate student's laptop. We collected both HTTP traces and user's inputs (keyboard and mouse events) for a 30-minute session. The test data contains a total of 3724 HTTP requests, including 24 update requests and 33 vagabonds (12 abnormal and 21 misconfigured requests).

5.1 Analysis of User Preference

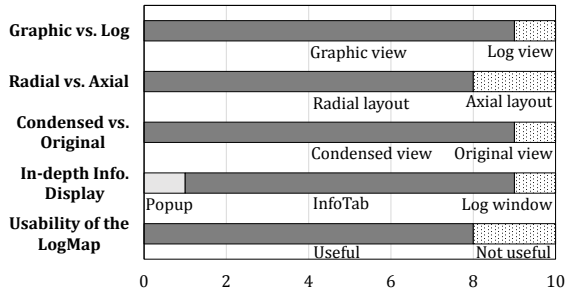


Figure 5: The breakdown of results on user preference in the user study.

Figure 5 presents a brief summary about user's preferences of major features in ReView. All ten participants correctly identified root-triggers, the first level of network requests, and answered the exact maximum level of requests. Based on results shown in Figure 5, nine participants indicated that they preferred visualizations over traditional network logs. They agreed on the fact that the provided visualizations in ReView summarize the network data, compared with the overwhelming information in raw logs. With ReView, users can quickly understand and make use of the structure of network causal relationship for security analysis.

ReView effectively navigates users among a large amount of network requests through different levels of abstractions. Eight participants preferred the radial layout over the axial one, because it enables the efficient utilization of the screen space. In addition, participants choosing the radial layout mentioned that they can navigate the display with minimal mouse scrolling needed. Other participants prefer the axial layout because they think the forest-like design is more salient and intuitive.

Nine participants preferred the condensed view over the original one. They agreed that the condensed view simplified representations by avoiding unnecessary leaf nodes. One participant who was in favor of the original view also admitted that condensed view is neat, but he still chose original view in case of missing necessary information. Last, eight participants found `LogMap` useful for showing the network requests by their distributions over time.

5.2 Analysis of Suspicious Requests

Nine participants identified the exact number of vagabond requests (33 nodes in total). The other one participant misunderstood the task and counted the vagabonds on the first level. All participants correctly listed detailed information of examples corresponding to each type of the vagabond. They were able to distinguish among different types of vagabond requests by digesting the request information using our tool.

With some domain knowledge, participants can infer reasons for the occurrences of malicious requests. For example, 8 participants answered that the first malicious request was sent to `spi.domainsponsor.com` and the domain was blacklisted as spyware. By reviewing its long URL string, participants found out that the request URL contains some substrings, such as `migTrackDataExt` and `migAgencyId`. Therefore, they speculated the outbound request is used to leak host

information. Users also found that the display of triggering relations for all requests benefits the analysis of malicious ones. For example, participant #9 pointed out that a similar vagabond request is sent out after a website is visited twice, so the website hosts may be compromised by malware or due to misconfiguration.

`LogMap` also helps to direct users to potentially suspicious network traffics with color coding of the frame for each grid. Participants #4, #7 and #10 regarded this as an important factor as to why they preferred `LogMap`. They used the tool to find out that the update requests were often sent out during the idle time, while abnormal ones were sent out along with some legitimate requests.

6. RELATED WORK

The dependency analysis provides an effective way to pinpoint the origin of the vulnerability and reason the complex structures (e.g., finding the dependency of network services [4] and intrusion detection logs [12], detecting drive-by download attacks [20]). However, the dependency analysis on network requests has not been well studied, with a few exceptions [14, 19]. WebProphet [14] extracts the dependencies between web objects in the light of their delays. ReSurf [19] reconstructs web surfing activities from network traces based on a referrer-based inference.

A comparative study from Goodall [9] confirmed that visualizations, compared with traditional logs, can help users easily perceive patterns and anomalies for cyber security related analysis. Many visualization designs have been proposed as intuitive and efficient means of displaying complex structures that cannot be explicitly identified in the raw data [5, 17]. Radial Traffic Analyzer [11] and Traffic Circle [2] are two network visualization tools that adopt the radial layout. Both solutions are used for monitoring the network (e.g., identifying communication partners and traffic types) by displaying the raw packet or flow information. In ReView, we use concentric arcs to represent different dependency levels, in such way that our tool provides logical relation of the network traffic in an information-rich fashion.

In recent years, many research efforts have been dedicated to visualize the network traffic, as visualization tools help researchers understand the structure of data [17,22] and identify the anomalies [1,10]. Traffic Causality Graph (TCG) [1] enables the profiling of network application through the temporal and spatial causality of flows. This solution aggregates the packets into flows and focuses on the causality of them. The causality finding algorithm in [1] is based on the protocol, IP and port information of flows, while ours is based on higher level application layer information, which is a fine-grained model and has better descriptive power. Traffic Dispersion Graph (TDG) [10] is used to present the network-wide flows by aggregating the packets. TDG captures the interaction among hosts in a network, while our TRG is on the host-based network traffic. Last, our model differs from TCG and TDG, as their solutions aim at classifying the traffic or detecting port-related threats (e.g., port scanning). However, our TRG is for identifying the abnormal requests through revealing the triggering relations of outbound packets.

7. CONCLUSIONS

Discovering traffic dependency has been shown to be an effective way to analyze network activities and identify malicious events. We introduced a new concept of high visual locality and developed ReView, a visualization tool that maximizes the usage of screen and helps security analysts better utilize network traffic dependency. In our design, we adopted a radial layout that supports to high visual locality. ReView serves as an integrated solution for analysts to accurately pinpoint anomalous network events and perform further investigation. User study results suggest that our tool provides usable visualization and interaction to help interpret network traffic causality and enhance security analysis.

8. REFERENCES

[1] H. Asai, K. Fukuda, and H. Esaki. Traffic Causality Graphs: Profiling network applications through temporal and spatial causality of flows. In *ITC'11*, pages 95–102, 2011.

[2] D. M. Best, S. Bohn, D. Love, A. Wynne, and W. A. Pike. Real-time visualization of network behaviors for situational awareness. In *VizSec'10*, pages 79–90, 2010.

[3] A. Boschetti, L. Salgarelli, C. Muelder, and K.-L. Ma. TVi: a visual querying system for network monitoring and anomaly detection. In *VizSec'11*, page 1, 2011.

[4] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *OSDI'08*, pages 117–130.

[5] G. Conti, K. Abdullah, J. Grizzard, J. Stasko, J. A. Copeland, M. Ahamad, H. L. Owen, and C. Lee. Countering security information overload through alert and packet visualization. *Computer Graphics & Applications, IEEE*, 26(2):60–70, 2006.

[6] D3.js: a JavaScript library to display given digital data into graphic, dynamic forms. `http://d3js.org`.

[7] G. A. Fink, P. Muessig, and C. North. Visual correlation of host processes and network traffic. In *VizSec'05*, page 2, 2005.

[8] J. Fonseca, M. Vieira, and H. Madeira. Vulnerability & attack injection for web applications. In *DSN'09*, pages 93–102, 2009.

[9] J. R. Goodall. Visualization is better! A comparative evaluation. In *VizSec'09*, pages 57–68, 2009.

[10] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *IMC'07*, pages 315–320, 2007.

[11] D. A. Keim, F. Mansmann, J. Schneidewind, and T. Schreck. Monitoring network traffic with radial traffic analyzer. In *VAST'06*, pages 123–128, 2006.

[12] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen. Enriching intrusion alerts through multi-host causality. In *NDSS'05*, 2005.

[13] H. Koike and K. Ohno. SnortView: Visualization system of Snort logs. In *VizSEC/DMSEC'04*, pages 143–147, 2004.

[14] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang. WebProphet: Automating performance prediction for web services. In *NSDI'10*, pages 143–158, 2010.

[15] Malware domain list. `http://www.malwaredomainlist.com`.

[16] Suspicious domains from SANS institute. `https://isc.sans.edu/suspicious_domains.html`.

[17] H. Shiravi, A. Shiravi, and A. A. Ghorbani. A survey of visualization systems for network security. *IEEE Trans. Vis. Comput. Graph.*, 18(8):1313–1329, 2012.

[18] H. Song, C. Muelder, and K.-L. Ma. Crucial nodes centric visual monitoring and analysis of computer networks. In *CyberSecurity*, pages 16–23, 2012.

[19] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, and Y. Jin. ReSurf: Reconstructing web-surfing activity from network traffic. In *IFIP Networking Conference*, pages 1–9, 2013.

[20] K. Xu, D. Yao, Q. Ma, and A. Crowell. Detecting infection onset with behavior-based policies. In *NSS'11*, pages 57–64.

[21] L. Xu, Z. Zhan, S. Xu, and K. Ye. Cross-layer detection of malicious websites. In *CODASPY'13*, pages 141–152, 2013.

[22] W. Yu and R. M. Verma. Visualization of rule-based programming. In *SAC'08*, pages 1258–1259, 2008.

[23] ZeuS tracker domain blocklist. `https://zeustracker.abuse.ch/blocklist.php`.

[24] H. Zhang, W. Banick, D. Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 104–112, 2012.

[25] H. Zhang, D. Yao, and N. Ramakrishnan. Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In *ASIACCS'14*, pages 39–50, 2014.