Hviz: HTTP(S) Traffic Aggregation and Visualization for Network Forensics

David Gugelmann[a,*], Fabian Gasser[a], Bernhard Ager[a], Vincent Lenders[b]

[a] *ETH Zurich, Zurich, Switzerland*
[b] *armasuisse, Thun, Switzerland*

Abstract

HTTP and HTTPS traffic recorded at the perimeter of an organization is an exhaustive data source for the forensic investigation of security incidents. However, due to the nested nature of today's Web page structures, it is a huge manual effort to tell apart benign traffic caused by regular user browsing from malicious traffic that relates to malware or insider threats. We present Hviz, an interactive visualization approach to represent the event timeline of HTTP and HTTPS activities of a workstation in a comprehensible manner. Hviz facilitates incident investigation by structuring, aggregating, and correlating HTTP events between workstations in order to reduce the number of events that are exposed to an investigator while preserving the big picture. We have implemented a prototype system and have used it to evaluate its utility using synthetic and real-world HTTP traces from a campus network. Our results show that Hviz is able to significantly reduce the number of user browsing events that need to be exposed to an investigator by distilling the structural properties of HTTP traffic, thus simplifying the examination of malicious activities that arise from malware traffic or insider threats.

Keywords: network forensics, HTTP(S), event reconstruction, aggregation, visualization, incident investigation

1. Introduction

Network traces are one of the most exhaustive data sources for the forensic investigation of computer security incidents such as online fraud, cyber crime, or data leakage. By observing the network traffic between an internal network and the outside world, an investigator can often reconstruct the entire event chain of computer security breaches, helping to understand the root cause of an incident and to identify the liable parties. In particular, the investigation of HTTP traffic is becoming increasingly important in digital forensics as HTTP has established itself as the main protocol in corporate networks for client-to-server communication [1]. At the same time, malware, botnets and other types of malicious activities nowadays extensively rely on HTTP communication [2], possibly motivated by the ubiquitous access to the Web even in locations where Internet access is otherwise strictly policed.

Manually analyzing HTTP traffic without supportive tools is a daunting task. Traffic of a single workstation can easily account for millions of packets per day. Even when the individual packets of an HTTP session are reassembled, the traffic may exhibit an abundant number of requests. This high number of requests results from how Web pages are built today. When a browser first loads a Web page from a server, dozens to hundreds of additional HTTP requests are triggered to download further content, such as pictures [3, 4]. These requests may be addressed to the same server as the original page. However, today's common practice of including remote elements, such as advertisements or images hosted on CDNs, results in numerous requests to third-party servers as well. Consequently, finding suspicious activities in a network trace oftentimes resembles the search for a needle in a haystack.

We present Hviz (HTTP(S) traffic visualizer), a traffic analyzer that reconstructs and visualizes the HTTP and HTTPS traffic of individual hosts. Our approach facilitates digital forensics by *structuring, aggregating,* and *correlating* HTTP traffic in order to reduce the number of events that need to be exposed to the investigator.

Hviz reduces the number of HTTP events by combining data aggregation methods based on frequent item set mining [5] and domain name based grouping with heuristics to identify main pages in HTTP requests [6, 7]. To support the investigator at finding traffic anomalies, Hviz further exploits cross-computer correlations by highlighting traffic patterns that are unique to specific workstations. Hviz visualizes the aggregated events using a JavaScript based application running in the Web browser.

Our main contributions are the following:

- We propose an approach for grouping and aggregating HTTP traffic into abstract events which help understanding the structure and root cause of HTTP requests issued by individual workstations.
- We present Hviz, an interactive visualization tool based on the proposed approach to represent the event timeline of HTTP traffic and explore anomalies based on cross-computer correlation analysis.
- We evaluate the performance of our approach with synthetic and real-world HTTP traces.

*Corresponding author
Email address: gugelmann@tik.ee.ethz.ch (David Gugelmann)

As input data, Hviz supports HTTP and HTTPS traffic recorded by a proxy server [8] and HTTP network traces in tcpdump/libpcap format [9]. We make Hviz's interactive visualization of sample traces available at `http://hviz.gugelmann.com`.

In the remainder of this paper, we formulate the problem in Sect. 2 and introduce our design goals and core concepts in Sect. 3. In Sect. 4, we present our aggregation and visualization approach Hviz. We evaluate our approach in Sect. 5 and discuss evasion strategies and countermeasures in Sect. 6. We conclude with related work in Sect. 7 and a summary in Sect. 8.

2. Problem Statement

When a security administrator receives intrusion reports, virus alerts, or hints about suspicious activities, he may want to investigate the network traffic of the corresponding workstations in order to better understand whether those reports relate to security breaches. With the prevalence of Web traffic in today's organization networks [1], administrators are often forced to dig into the details of the HTTP protocol in order to assess the trustworthiness of network flows. However, Web pages may exhibit hundreds of embedded objects such as images, videos, or JavaScript code. This results in a large number of individual HTTP requests each time a user visits a new Web page.

As an example, during our tests, we found that users browsing on news sites cause on average more than 110 requests per visited page. Even more problematic than the mere number of requests is the fact that on average a single page visit resulted in requests to more than 20 different domains. These numbers, which are in line with prior work [3, 4], clearly highlight that manually analyzing and reconstructing Web browsing activity from network traces is a complex task that can be very time-consuming.

Malicious actors can take advantage of this issue: Recent analyses of malware and botnet traffic have shown that the HTTP protocol is often used by such actors to issue command and control (C&C) traffic and exfiltrate stolen data and credentials [2]. Our aim is therefore to support an investigator at investigating the HTTP activity of a workstation when looking for malicious activities, such that

1. the investigator can quickly understand which Web sites a user has visited and
2. recognize malicious activity. In particular, HTTP activity that is unrelated to user Web browsing such as malware C&C-traffic should stand out in the visualization despite the large amount of requests generated during Web browsing.

3. Design Goals and Concepts

We start this section by introducing our terminology. Then, we present the underlying design goals and describe the three core concepts behind Hviz.

3.1. Terminology

For simplicity we use the term HTTP to refer to both HTTP and HTTPS, unless otherwise specified. We borrow some of our terminology from ReSurf [7]. In particular, a *user request* is an action taken by a user that triggers one or more HTTP requests, e.g., a click on a hyperlink or entering a URL in the address bar. The first HTTP request caused by a user request is referred to as the *head request*, the remaining requests are *embedded requests*. We refer to a request that is neither a head nor an embedded request as *other request*. These requests are typically generated by automated processes such as update services or malware. The sequence of *head requests* is the *click stream* [10]. We organize HTTP requests of a workstation in the *request graph*, a directed graph with HTTP requests as nodes and edges pointing from the Referer node to the request node (see Sect. 4.1.1 for details).

3.2. Design Goals

The aim of Hviz is to visualize HTTP activity of a workstation for analysis using input data recorded by a proxy server or network gateway. In particular, Hviz is built according to the following design goals:

I. Visualize the timeline of Web browsing activity (the *click stream*) of a workstation such that an investigator can quickly understand which Web pages a user has visited.
II. Support an investigator in understanding why a particular service receives HTTP requests, i.e., if the service receives requests as part of regular Web browsing or because of a suspected attack or data exfiltration.
III. Reduce the number of displayed events to avoid occlusion.
IV. Prevent HTTP activity from getting lost in the shuffle. For example, a single request to a malware C&C server should be visible among hundreds of requests caused by regular Web browsing.

3.3. Core Concepts

Hviz relies on three core concepts:

1. To achieve design goal I, we organize HTTP requests in the request graph and apply a heuristic [7] to distinguish between requests that are directly triggered by the user (head requests) and requests happening as a side effect (embedded requests). The sequence of head requests visualized in chronological order provides the "big picture" of Web browsing activity. The graph helps the understanding of how a user arrived at a Web page (design goal II).
2. It might be tempting to reduce the visualization to head requests in order to achieve the reduction of events demanded by design goal III. However, this approach comes with three drawbacks: *(i)* Typical malware causes HTTP requests that are unrelated to Web browsing and, as a consequence, would disappear from the visualization (conflict with design goal IV). *(ii)* Knowing how head requests are identified, an attacker can intentionally shape

his HTTP activity such that malicious activities are missed (conflict with design goal II). *(iii)* Incorrectly classified HTTP requests become difficult to recognize and understand without the related HTTP requests (conflict with design goal IV). Instead of completely dropping non-head requests, we reduce the number of visualized events by means of domain aggregation and grouping based on frequent item set mining. This way, the number of visualized events is reduced (design goal III), while HTTP events that are not part of regular Web browsing are still visible (design goal IV).

3. To help decide if a request is part of regular Web browsing, a suspected attack against a workstation, or data exfiltration (design goal II), Hviz correlates the HTTP activity of the host under investigation with the activity of other workstations in the network. HTTP requests that are similar to requests issued by other workstations can be faded out or highlighted interactively.

4. Hviz

Hviz uses several data processing steps to achieve its goals of reducing the number of visualized events and highlighting important activity. We describe these processing steps in this section. Further, we introduce and explain our choices for visualization.

4.1. Input Data and Architecture

Hviz can either operate on network packet traces or on proxy log files. Packet traces are simple to record. However, in packet traces it is typically not possible to access the content of encrypted HTTPS connections. Thus, in a high-security environment, an intercepting HTTP proxy enabling clear-text logging of both HTTP and HTTPS messages may be preferable. Making the use of the proxy mandatory forces potential malware to expose their traffic patterns.

Hviz currently supports HTTP and HTTPS traffic recorded by the mitmdump proxy server [8] and HTTP traffic recorded in tcpdump/libpcap format [9]. We use a custom policy for the Bro IDS [11] to extract HTTP messages from libpcap traces.

The architecture of Hviz consists of a preprocessor and an interactive visualization interface. The preprocessor—a Python program—runs on the server where the HTTP log data is stored. The visualization interface uses the D3 JavaScript library [12] and runs in the Web browser. We provide an overview of the processing steps in Figure 1, and explain each of the steps in the rest of this section.

4.1.1. Building the Request Graph

As a first step, Hviz analyses the causality between HTTP requests, represented by the *request graph*. Figure 1 illustrates the process in box (A). Each node in the request graph represents an HTTP request and the corresponding HTTP response. If an HTTP request has a valid Referer header, we add a directed edge from the node corresponding to the Referer header to the node corresponding to the HTTP request. For example,

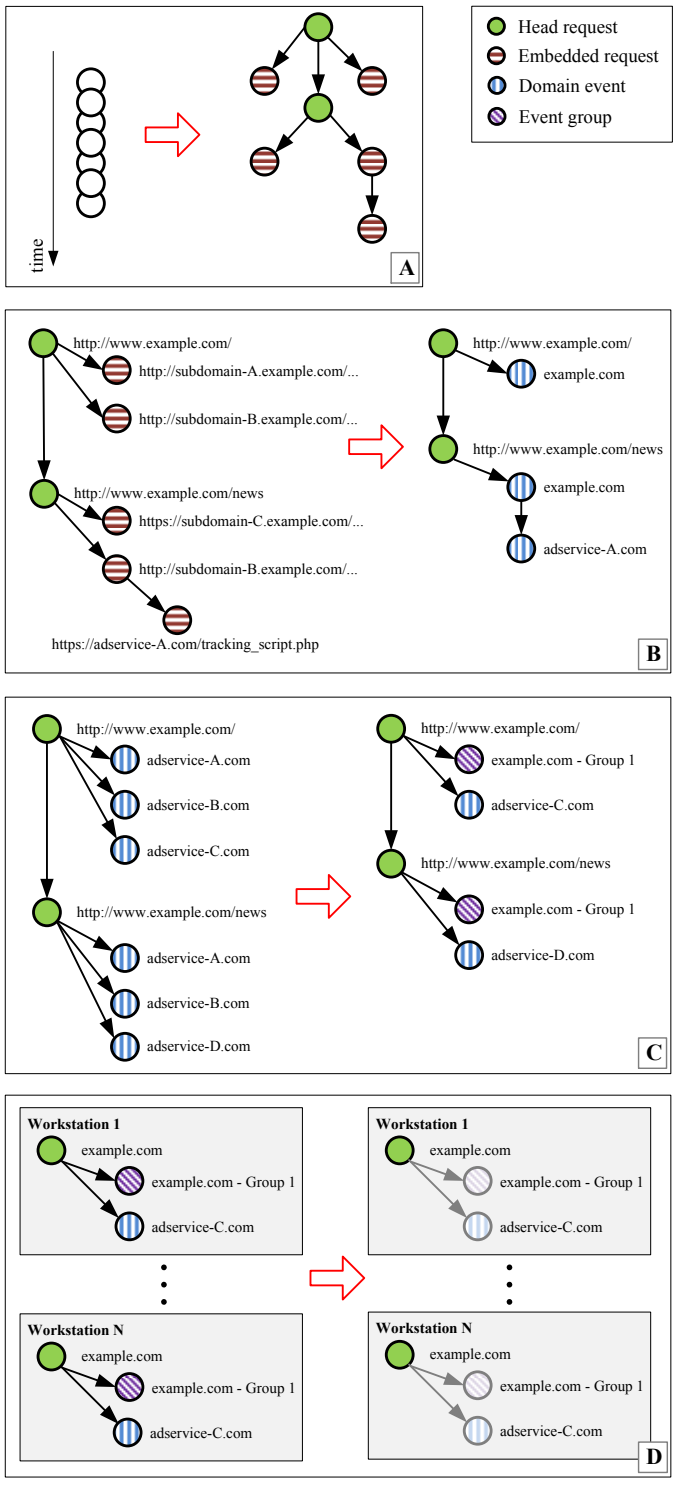

Figure 1: Schematic visualization of the processing steps in Hviz: (A) Reconstruction of request graph from HTTP requests; (B) aggregation of embedded requests to domain events; (C) aggregation of domain events to meta events; (D) correlation between workstations to identify and fade out popular events.

when a user is on `http://www.bbc.com` and clicks a link leading him to `http://www.bbc.com/weather`, the HTTP request for the weather page contains `http://www.bbc.com` in the Referer header. In this case, we add a directed edge from `http://www.bbc.com` to `http://www.bbc.com/weather` to the graph.

Requests for embedded objects that are issued without user involvement, e.g., images, usually also contain a Referer header. To tell apart head requests (requests that are directly triggered by the user) from embedded requests (requests for embedded objects), Hviz relies on the ReSurf heuristic [7]. Hviz tags the identified head nodes and memorizes their request times for later processing steps.

4.1.2. Event Aggregation

The sheer number of HTTP requests involved in visiting just a handful of Web sites makes it difficult to achieve a high-level understanding of the involved activities. Thus, we need to reduce the number of displayed events by dropping, or by aggregating similar events. Dropping, however, would violate design goal IV (see Sect. 3.2). For example, only displaying head requests would render the C&C traffic traffic caused by the Zeus spyware [13] undetectable, because the corresponding requests are not (and should not be) classified as head requests by ReSurf [7].

As a consequence, we rely on aggregation for the visualization purpose, and provide access to the details of every request on user-demand. As a first step, we visualize embedded requests at the granularity of domains. Specifically, we aggregate on the effective second level domain[1]. For example, as shown in box (B) in Figure 1, embedded requests to `subdomain-A.example.com` and `subdomain-B.example.com` are summarized to one *domain event* with the effective second level domain `example.com`.

Nearly all Web sites include content from third parties, such as CDNs for static content, advertisement and analytics services, and social network sites. As a result, embedded objects are often loaded from dozens of domains when a user browses on a Web site. Such events cannot be aggregated on the domain level. However, the involved third-party domains are often the same for the different pages on a Web site. That is, when a user browses on `example.com` and embedded objects trigger requests to the third parties `adservice-A.com`, `adservice-B.com` and `adservice-C.com`, it is likely that also other pages on `example.com` will include elements from these third parties. We use this property to further reduce the number of visualized events by grouping domain events that frequently appear together as *event groups*. Figure 1 illustrates this step in box (C). To identify event groups, we collect all domain events triggered by head requests from the same domain and group these domains using frequent item set mining (FIM) [5].

This approach may suppress continuous activity towards a single domain or domain group. Therefore, our approach additionally ensures that HTTP requests which occur more than five minutes apart are never grouped together. As a result, requests which are repeated over long periods of time appear as multiple domain events or multiple event groups and can be identified by inspection.

4.1.3. Tagging Popular and Special Events

By only looking at the visited URL or domain name, it is often difficult to tell if a request is part of regular Web browsing or a suspected malicious activity. To help with this decision, we introduce tagging of events.

Hviz correlates the HTTP activity of multiple workstations to determine the popularity of events. If an activity is popular (i.e., seen in the traffic of many workstations), one should assume that it is regular Web browsing and, as such, probably of little interest. We measure the popularity of an event by counting on how many workstations we see requests to the same domain with the same Referer domain, i.e., the same edge in the request graph. For example, in box (D) in Figure 1, we calculate the popularity of the domain event `adservice-C.com`, by counting on how many workstations we see an edge from `example.com` to `adservice-C.com`. If this edge is popular, it is most likely harmless. We tag a node as popular if the popularity of all incoming edges (event groups can have multiple incoming edges) is greater than or equal to a threshold. The threshold can be interactively adjusted in Hviz.

Similarly, we tag nodes that may be of special interest during an investigation. We tag file uploads because uploads can be hints on leakage of sensitive information. Nodes with upload data are never aggregated to event groups and not tagged as popular. In addition, the uploaded payload is reassembled and made available in the visualization. For demonstration purposes we limit ourselves to file uploads. However, the tagging system is extensible. In the future, we intend to incorporate additional information sources such as Google Safe Browsing[2], abuse.ch, or DNS-BH[3].

4.2. Visualization

The browser-based visualization interface of Hviz consists of a main window and an optional pop-up window showing HTTP request details (see Figure 2). The main window shows the visualization of the reduced request graph and a panel on the left with additional information. At the top of the window we provide two boxes with visualization controls.

Events are displayed as nodes, and the Referer relationship between events corresponds to directed edges. The size of nodes is proportional to the outgoing HTTP volume (plus a constant). Hviz fades out the (probably innocuous) popular nodes to reduce their visual impact. Head requests are visualized as green nodes and placed along the vertical axis in the order of arrival. This enables the investigator to follow the click

[1] `http://publicsuffix.org/`

[2] `https://developers.google.com/safe-browsing/`

[3] `http://www.malwaredomains.com/`

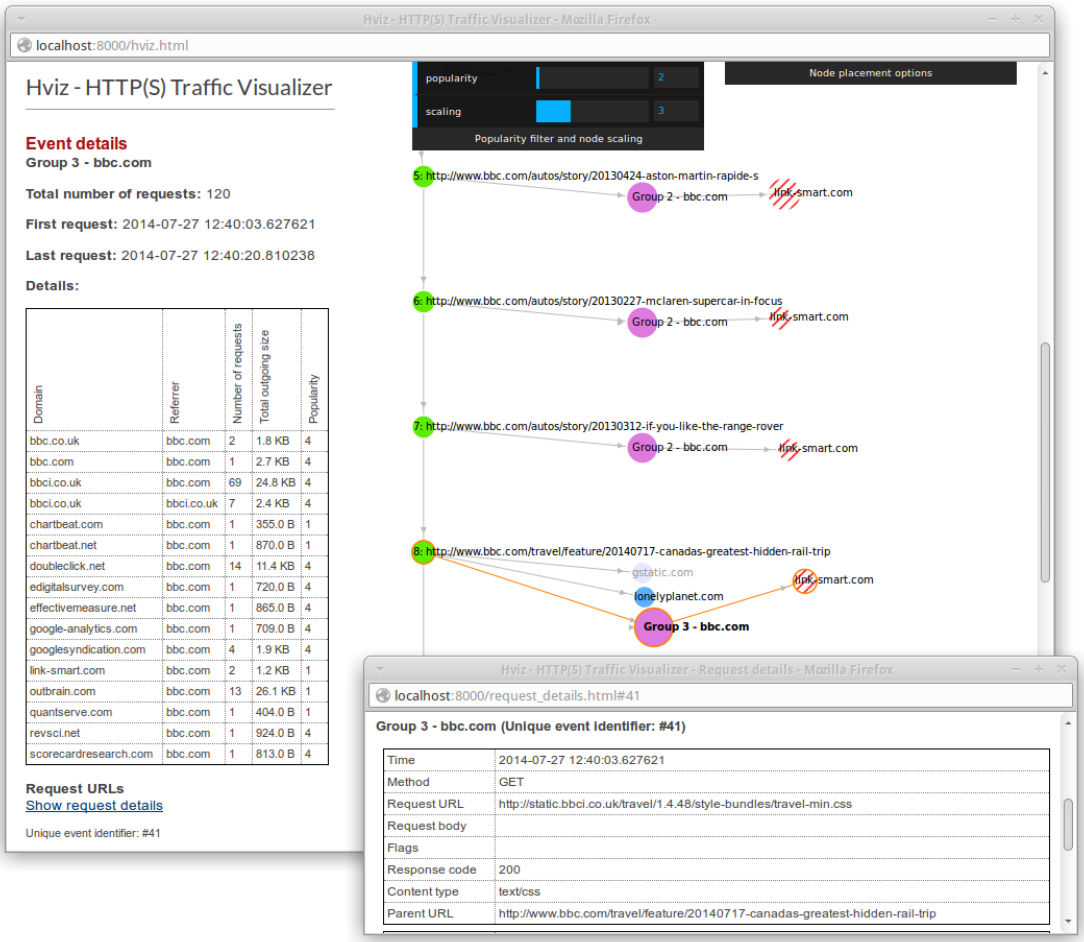

Figure 2: Screenshot of Hviz visualizing Web browsing activity of an author of this paper. The main window shows the click stream and event summaries. The smaller window shows HTTP request details and allows to search the content. The click stream is visualized as a graph in the main window. Head requests (green nodes) are ordered by time on the y-axis. Groups of embedded domains (purple nodes) and single domains (blue nodes) branch to the right. The size of the nodes is proportional to the outgoing HTTP volume (plus a constant). The node size scaling factor can be interactively adapted by using the slider at the top of the window. The second slider at the top adjusts the popularity threshold used to fade out nodes. Data uploads are marked with red hatches.

stream by simply scrolling down. To keep the visualization compact and well-structured, embedded events are branching to the right, independent from their request times. Domain groups are displayed in purple color, domain events in blue. Other requests without Referer, e.g., software updates triggered by the operating system or malware requests, get the color yellow (see Figure 5). Hviz highlights special events. In particular, to make uploads stand out because of their importance in data exfiltration, Hviz displays HTTP requests containing a body using red hatches.

Hviz initially assigns a fixed position to head nodes and their first hop children. For positioning of second hop and higher children, we rely on D3's force-directed layout [12]. At any time, an investigator can toggle the positioning of a node between force-layout and fixed position with a double-click, or move a node around to improve the choices of the automated layout.

The panel on the left displays information on the currently selected event, such as the names of the involved request and Referer domains and the total number of requests represented by an event. The two boxes at the top provide sliders for the node size scaling, popularity threshold, and control of the force layout, e.g., adjusting the amount of force pulling floating nodes to the right. When a user clicks on "Show request details" in the left panel, a pop-up window appears providing further details on the currently selected event. This includes the timestamp, request method, URL, and parent URL. Hviz reassembles file uploads and makes them available in the pop-up window as well. Because the pop-up window shows a single document that is linked to Hviz's main window using HTML anchor tags, the pop-up window can as well be used for free text search when looking for specific events.

5. Evaluation and Usage Scenarios

In this section, we investigate how powerful Hviz's visualization is. We find improved parameters for the ReSurf [7] heuristic, and examine how much aggregation and popularity filter can help in reducing the number of events. We discuss

Parameter name	ReSurf	Hviz
$min\_response\_length$	3000	3000
min_time_gap	0.5	3
$min\_referrals$	2	2
$head\_as\_parent$	True	False

Performance	ReSurf	Hviz
Recall	80 %	81 %
Precision	75 %	90 %
F1-measure	78 %	86 %

Table 1: Parameters and head node detection performance.

scalability and conclude this section with showcases demonstrating how Hviz handles specific incidents.

5.1. Head Node Detection Accuracy

Correctly identified head nodes greatly support an investigator during analysis. Thus a high detection accuracy is desirable. In order to understand the influence of parameters on the ReSurf [7] heuristic, we perform a parameter investigation.

For privacy reasons, we rely on synthetic ground truth traces. We create these traces using the Firefox Web browser and a browser automation suite called Selenium [14]. We instruct Firefox to visit the 300 most popular Web sites according to Alexa[4] as of July 2014. Starting on each landing page, the browser randomly follows a limited number of links which reside on the same domain as the landing page. The number of links is selected from a geometric distribution with mean 5, and the page stay time distribution is logarithmic normal[5] with $\mu = 2.46$ and $\sigma = 1.39$, approximating the data reported by Herder [15]. We note that not all of the top sites are useful for our purposes, for two reasons. *(i)* Some of the sites do not provide a large enough number of links, e.g., because they are entirely personalized. *(ii)* As we detected later, some sites can only be browsed via the HTTPS protocol, yet we only recorded packet traces. Still, in total, our dataset covers the equivalent of 1.3k user requests and contains 74k HTTP requests.

We perform parameter exploration in order to optimize the detection performance. In Table 1, we summarize our findings, and Figure 3 shows the recall and precision values achieved for different parameters. The difference regarding min_time_gap may result from differences in the utilized traces, or from the way that the time gap is measured. Unfortunately ReSurf does not exactly specify at which time the gap starts. We achieve the highest F1-measure with $head\_as\_parent = False$. As a consequence head nodes are detected independently from each other. In contrast, if using the original ReSurf configuration, missing a head node would cause all following head nodes in the request graph to remain undetected too.

5.2. Aggregation Performance

The main purpose of Hviz is to assist in understanding the relations in HTTP traffic. For an investigator, what matters is the time he spends on getting an accurate understanding. As this time is difficult to assess, we instead rely on the number of events that need to be inspected as an indicator for the time an investigator would have to invest.

[4] `http://www.alexa.com/topsites/countries/CH`

[5] Note that μ and σ describe the underlying normal distribution.

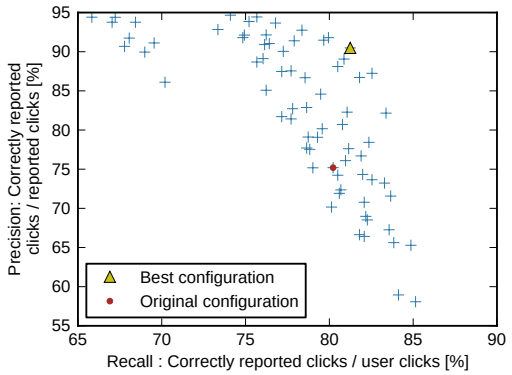

Figure 3: Head node detection performance. Every marker is a run with different parameters. The original ReSurf configuration is shown as dot, the configuration with highest F1-measure as triangle.

5.2.1. Dataset

We collected TCP port 80 traffic of 1.8k clients in a university network over a period of 24 hours. In total, this corresponds to 205 GB of download traffic and 7.4 GB of upload traffic from 5.7M HTTP requests. 1.0k of the clients contact at least 50 different Web servers. We randomly select 100 of these clients and measure how Hviz would perform during an investigation of their HTTP activity. Within this set of 100 clients, the median client issued 36 head requests and triggered 2.4k HTTP requests in total. To protect user privacy we refrain from visualizing HTTP activity based on this dataset, but limit ourselves to producing aggregated statistics.

5.2.2. Domain- and FIM-based Grouping

Hviz groups HTTP requests to domain events and further aggregates these events using frequent item set mining (FIM). We calculate the reduction factor for these steps as the number of all HTTP requests issued by an IP address divided by the number of events remaining after grouping. Figure 4 shows the results for our 100 sample clients. We achieve a 7.5 times reduction in the median, yet the factors for the individual clients range from 3 to more than 100.

5.2.3. Popularity-based Filtering

As next step, we evaluate the effect of Hviz's popularity filter. This filter identifies, with the granularity of SLDs, popular referrals, i.e., when (Referer domain, request domain)-pairs are originated from many hosts. We deem these events most likely innocuous, and, as a consequence, these events are tagged as *popular* events and faded out in the visualization (see Sect. 4.1.3). For this analysis, we set the popularity filter threshold to 10. We then calculate the reduction factor as the number of all HTTP requests issued by a client divided by the number of HTTP requests that are not tagged as popular. The reduction factor for our 100 sample clients is displayed in Figure 4. The median reduction factor is 2.9. Interestingly, a small number of clients does barely benefit from popularity-based filtering, indicating special interests.

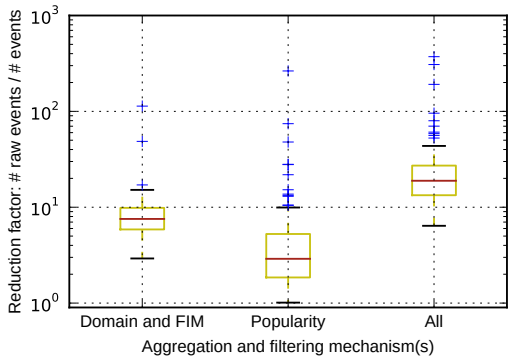

Figure 4: Filtering and aggregation performance as box plots. The red lines in the boxes represent the medians.

5.2.4. Overall Effectiveness of Hviz

We use the term *active events* to refer to the (not faded out) events remaining after applying domain- and FIM-based grouping, and popularity filtering. Again, we choose 10 as the threshold for the popularity filter. We calculate the overall reduction factor as the number of all HTTP requests issued by a client divided by the number of active events. Overall, Hviz achieves a 18.9 times reduction. Figure 4 shows a box plot of the distribution over the 100 sample clients.

For most clients, domain- and FIM-based grouping is more effective than applying the popularity filter. For example, we found one client which extensively communicated with a single, unpopular service. In this case, the popularity filter is almost ineffective. Yet, since these requests are targeted to the same domain they can be very well grouped. Overall, the number of HTTP requests of this client is more than 190fold higher than its number of active events.

We also have evidence of the opposite, i.e., the popularity filter being highly effective yet domain- and FIM-based grouping not working well. For example, one client issued almost all requests to a variety of popular services. Popularity filtering therefore reduces the number of events by almost a factor of 50.

When comparing grouping with popularity reduction factors we find no correlation, thus we infer that these two reduction methods work (largely) independently. Considering all 100 clients, the median 2.4k raw HTTP requests are reduced to a far more approachable 135 active events per client. The median reduction factor is 18.9.

5.2.5. Scalability

We evaluate scalability according to two criteria, *(i)* the time required to prepare data for visualization, and *(ii)* the interactivity of the visualization. In order to estimate the scalability of the data processing step, we measure the processing time when analyzing the above dataset. The dataset in pcap format includes 212 GB of HTTP traffic in total, covering 24 hours of network activity of 1.8k users. Processing is CPU-bound. Running on one CPU of an Intel Xeon E5-2670 Processor, it takes 4 hours to extract HTTP requests and responses. Building and analyzing the request graphs for the 100 analyzed clients from the preprocessed data takes 30 minutes. We conclude that the data processing scales up to thousands of clients.

Figure 5: Hviz visualizing Zeus trojan activity taking place during regular Web browsing. The C&C server of this Zeus variant was located at `greenvalleyholidayresort.com`.

To investigate the scalability of the visualization, we perform tests with artificial traces of incrementing size. Our experience shows that Hviz can handle graphs with up to 10k events before the interactivity of the display becomes sluggish. This corresponds to 5 times the number of events generated by the busiest client in the 24h trace. Visualizations containing more than 10k nodes can be split in the time domain into multiple smaller parts.

5.3. Use Cases

In this section, we give three examples to further illustrate how Hviz aggregates and visualizes malicious HTTP activity.

5.3.1. Visualization of Zeus Malware Activity

The Zeus malware family belongs to the most popular trojans specialized on stealing credentials [2]. As a first use case, we show how Hviz handles the activity of a workstation infected with Zeus.

We synthesize an example trace by merging a Zeus traffic sample and a short sample of a Web browsing session. Figure 5 shows the visualization of the synthetic trace. The C&C server of this Zeus malware sample was located at `greenvalleyholidayresort.com`[6]. Zeus does not set fake Referer headers, i.e., Zeus does not attempt to pretend that its communication is part of regular Web browsing (see Sect. 6).

[6]The domain has been deleted since.

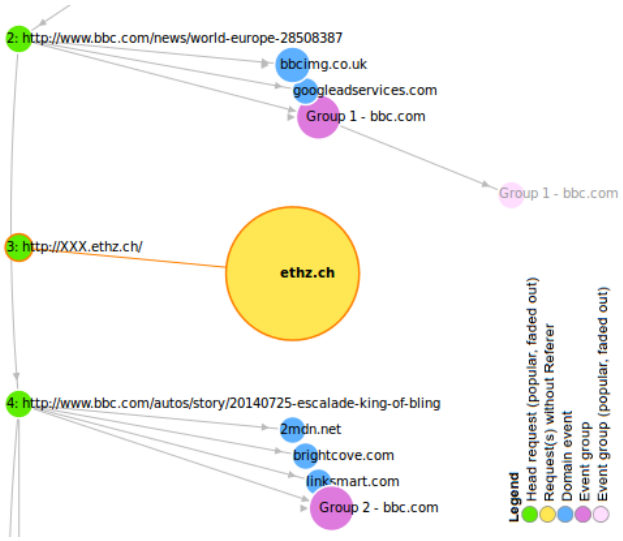

Figure 6: Hviz visualizing data leakage to a Web server via HTTP GET requests. The obfuscated upload clearly stands out as large node even though the total upload size is less than 2MB. The large node has an incoming edge and is still yellow because it groups requests with and without Referer. (The name of the server used for this experiment is anonymized in the screenshot.)

As a consequence, the Zeus bot's first request—a request for the bot configuration—is an unconnected yellow node. The following requests are used to exfiltrate data from the infected workstation to the C&C server. Hviz highlights the corresponding uploads using red hatches, enabling an investigator to spot these uploads. This trace additionally contains Windows update background traffic and background traffic to Google. Requests without Referer to `microsoft.com` and `google.com` occur on many workstations, that is why the popularity filter fades out these events.

5.3.2. *Visualization of Data Leakage*

In the second use case, we show that data leakage as small as a few megabytes become well visible in Hviz. The reason is that Hviz scales nodes according to outgoing traffic volume. To create a scenario that is more challenging than a simple file upload, we *(i)* use regular Web browsing as background noise during the data upload and *(ii)* obfuscate the upload by splitting the file into small chunks, and transmitting each of these chunks as URL parameter in a request of its own. The total upload volume is less than 2 MB. Most importantly, the splitting step prevents simple HTTP POST and request size detectors from triggering an alarm. This includes Hviz, which does not mark the node with red hatches as an upload.

Still, as demonstrated in Figure 6, the file upload becomes apparent due to the upload volume based sizing of nodes. Because all HTTP requests containing the uploaded data have been sent within a minute, Hviz aggregates these uploads into one single event which is rendered as a single large node. In order to avoid this aggregation, an attacker could distribute the requests over prolonged periods of time or over many different domains. However, in the visualization Hviz would create many smaller nodes. Dozens or even hundreds of singular events may again raise attention.

Figure 7: Hviz visualizing HTTP data of the DFRWS 2009 Forensics Challenge [16]. The displayed excerpt immediately shows that a user visited yahoo.com, searched for *mardi gras* and *mardi gras king cake*, and visited the found Web sites.

5.3.3. *Visualization of DFRWS 2009 Forensic Challenge*

As a third use case, we visualize a publicly available pcap trace file[7] from the DFRWS 2009 Forensics Challenge [16]. In short, the task of this challenge is to find evidence that a hacker named nssad had published "inappropriate" images of the Mardi Gras carnival event in New Orleans. The suspect claims he was not responsible for any transfer of data. The pcap trace file has been recorded during early surveillance of the subject. It contains more than 800 HTTP requests.

Within the data set, Hviz identifies 41 head nodes. In Figure 7, we show an excerpt of the visualization. We can instantly see search requests to Yahoo regarding the Mardi Gras event and the consequent visit of resulting Web sites. Given the consistent request graph with normal head nodes (green) and corresponding requests for embedded objects (blue) it appears plausible that the Web pages have been visited with a regular Web browser during normal browsing. In contrast, malware would in all likelihood not query for the embedded objects nor set appropriate Referer headers.

The visualized excerpt shows that the user first visited `yahoo.com`, searched for *mardi gras*, and then visited `mardigrasneworleans.com` by following a link on

[7] `http://www.dfrws.org/2009/challenge/imgs/nssal-capture-1.pcap.bz2`

yahoo.com. On this Web site, the user then navigated to `kingcakes.html`. Next, the user went back to `yahoo.com` and refined the search to *mardi gras king cake*. On the results page, the user then followed a link to `wikipedia.org`.

In short, based on Hviz's visualization an investigator can instantly see that *(i)* multiple Web sites related to Mardi Gras have been visited, *(ii)* these Web sites were most likely visited during regular Web browsing of a user and *(iii)* the user had been deliberately searching for these Web sites and did not arrive there by accident.

6. Evasion strategies and defense

The quality of the visualization in Hviz is dependent on the reliability of the head node classification heuristic. This means that a better heuristic can lead to better visualization results, as well as that an attacker can try to subvert the classification heuristic to complicate analysis of an incident. In this section, we discuss the consequences of head node misclassification, and their potential for attackers to hide their attack. For this discussion, we assume that all HTTP and HTTPS traffic of the attacker is available in clear-text. This can be achieved by the mandatory use of an intercepting Web proxy.

We start by taking a look at what happens when a head node is mis-classified. If a node is labeled as head node while it should not be labeled as such, it will appear in Hviz's visualization in the time-line on the left and be colored in green. Oftentimes, an investigator can spot these nodes based on hints in the displayed URL. In the opposite case—if a true head node is classified as non-head—there are two possible outcomes: *(i)* If the node has a valid Referer it is placed together with the other embedded nodes and groups. *(ii)* If the node does not have a valid Referer it is rendered in yellow. For both cases, these mis-classified nodes generally exhibit a larger than usual tree of child nodes and can thus be spotted as well. Currently, we entirely rely on the ReSurf heuristic [7] for head node classification. However, replacing ReSurf with any other (and possibly better) heuristic is trivial, should ReSurf ever turn out to be a limiting factor.

So which attack vectors does this open for malware[8]? HTTP requests from malware not setting a valid Referer header will appear as yellow nodes in the visualization (Sect. 5.3.1). Therefore, in order to hide, the malware has to forge valid Referer headers, e.g., by issuing an initial request to an innocuous Web site and further on utilizing this Web site's URL as Referer. In addition, to hide among the popular Web sites, malware has two options. *(i)* If the install base is large enough the malware is classified as popular on its own. An investigator can defend against this attack by using historic data for the popularity analysis. *(ii)* Malware can imitate request patterns of popular Web sites, i.e., the secondary-level domains (SLDs) of both Referer header and Host header have to be identical to those in popular HTTP requests. This can be achieved by, e.g., exploiting a popular Web site, or by crafting Host headers not related to the contacted IP addresses. Fake host headers can be mitigated by the mandatory use of a Web proxy, or by additional checks on the Host name-to-IP address relationship.

If a popular Web service is (mis-)used as C&C-channel Hviz may fade out the relevant communication using the popularity filter, provided that the service is popularly used inside the attacked network. However, all related communication data remains visible and available to the investigator. Depending on communication frequency, repeated access patterns may become apparent.

To sum up, the use of a Web proxy and preservation of historical popularity data help to prevent attackers from forging arbitrary requests and from hiding their communication. In addition, we want to emphasize that Hviz does never suppress data. Thus, even if there is no historical popularity data available, an investigator can still reconstruct an incident with Hviz.

7. Related Work

Analyzing and reconstructing user click streams: In order to understand user search activity, Qiu et al. introduced the term Referer tree as the sequence of all pages visited by a user [17]. In the heuristic utilized by Qiu et al., HTTP objects with content type text are characterized as user requests. Kammenhuber et al. introduced a state machine to identify the sequence of pages visited by a user, coined "clickstream" [10]. StreamStructure [6] and ReSurf [7] improve on prior work by developing heuristics that allow to distinguish more reliably between user requests and embedded requests, thus enabling analysis of today's complex Web sites. Our work utilizes the ReSurf heuristic [7]. ClickMiner [18] is a more involving approach that actively replays recorded HTTP traffic in an instrumented browser to reconstruct user requests. In contrast to [7, 18], our focus is on aggregation and visualization, not on detection of user requests. Indeed, any heuristic identifying user requests could be used by our visualization approach.

Detecting HTTP-based malware: Perdisci et al. [19, 20] target on the detection of malware with HTTP C&C-channels. BotFinder [21] uses a content-agnostic approach suitable to detect HTTP based malware. These approaches rely on machine learning the behavior of malware from sample traces. In contrast, Hviz identifies common and thus probably boring traffic patterns and makes these patterns less prominent in the visualization. As a consequence, traffic that is unique to a workstation becomes more pronounced in the visualization. Zhang et al. [22] and Burghouwt et al. [23] both organize HTTP requests in a graph and correlate the graph with user actions in order to detect requests issued by malware. While their approaches rely on recording user actions such as mouse clicks and keystrokes, Hviz operates on network traffic only.

Visualization of network activity: Most work on visualizing network activity aims at identifying anomalies and consequently investigates network traffic as a whole. Shiravi et al. present an overview of the existing large body of work in this context [24]. Our work is complementary to these approaches by providing a tool to understand the relationships in HTTP

[8] For brevity, we only use malware as example, but the same principles apply to any kind of attacker.

traffic of a single workstation. The main idea is to use an existing solution such as NetGrok [25] or AfterGlow [26] to identify suspicious workstations and then utilize Hviz to inspect the HTTP activity of that workstation in detail. NetWitness Visualize [27] displays transmitted files and data in an interactive timeline. Since this visualization focuses on showing the files contained in HTTP traffic, it is not suitable for exploring the dependencies between Web objects.

8. Summary and Future Work

We present our HTTP traffic analyzer Hviz. Hviz visualizes the timeline of HTTP and HTTPS activity of a workstation. To reduce the number of events displayed to an investigator, Hviz employs aggregation, frequent item set mining and cross-correlation between hosts. We show in our evaluation with HTTP traces of real users that Hviz displays 18.9 times fewer active events than when visualizing every HTTP request separately while still preserving key events that may relate to malware traffic or insider threats.

As future work, we plan to incorporate additional information for event tagging, such as Google Safe Browsing, and more details on uploads and downloads.

Acknowledgement

This work was partially supported by the Zurich Information Security and Privacy Center (ZISC). It represents the views of the authors.

References

[1] Palo Alto Networks, Re-Inventing Network Security to Safely Enable Applications, https://www.paloaltonetworks.com/resources/whitepapers/re-inventing-network-security.html (November 2012).

[2] Dell SecureWorks, Top Banking Botnets of 2013, http://www.secureworks.com/cyber-threat-intelligence/threats/top-banking-botnets-of-2013/ (March 2014).

[3] R. Pries, Z. Magyari, P. Tran-Gia, An HTTP Web Traffic Model Based on the Top One Million Visited Web Pages, in: Proc. EURO-NGI Conference on Next Generation Internet (NGI), 2012. doi:10.1109/NGI.2012.6252145.

[4] M. Butkiewicz, H. V. Madhyastha, V. Sekar, Understanding website complexity: Measurements, metrics, and implications, in: Proc. IMC, ACM, New York, NY, USA, 2011, pp. 313–328. doi:10.1145/2068816.2068846.

[5] C. Borgelt, Frequent item set mining, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2 (6) (2012) 437–456. doi:10.1002/widm.1074.

[6] S. Ihm, V. S. Pai, Towards Understanding Modern Web Traffic, in: Proc. IMC, ACM, New York, NY, USA, 2011, pp. 295–312. doi:10.1145/2068816.2068845.

[7] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos, Y. Jin, ReSurf: Reconstructing Web-Surfing Activity From Network Traffic, in: IFIP Networking Conference, 2013, pp. 1–9.

[8] A. Cortesi, M. Hils, mitmproxy: a man-in-the-middle proxy, http://mitmproxy.org, last visited: 2014-09-22.

[9] Tcpdump/Libpcap, http://www.tcpdump.org, Last visited: 2015-01-10.

[10] N. Kammenhuber, J. Luxenburger, A. Feldmann, G. Weikum, Web Search Clickstreams, in: Proc. IMC, 2006. doi:10.1145/1177080.1177110.

[11] V. Paxson, Bro: a System for Detecting Network Intruders in Real-Time, Computer Networks 31 (1999) 2435–2463. doi:10.1016/S1389-1286(99)00112-7.

[12] M. Bostock, V. Ogievetsky, J. Heer, D3: Data-Driven Documents, IEEE Transactions on Visualization and Computer Graphics (Proc. InfoVis) 17 (12) (2011) 2301–2309. doi:10.1109/TVCG.2011.185.

[13] D. Macdonald, D. Manky, Zeus: God of DIY Botnets, http://www.fortiguard.com/legacy/analysis/zeusanalysis.html, Last visited: 2014-07-30.

[14] SeleniumHQ - Browser Automation, http://www.seleniumhq.org/, Last visited: 2015-01-10.

[15] E. Herder, Forward, Back and Home Again: Analyzing User Behavior on the Web, Ph.D. thesis, University of Twente (2006).

[16] DFRWS 2009 Forensics Challenge, http://www.dfrws.org/2009/challenge/index.shtml, Last visited: 2014-09-15.

[17] F. Qiu, Z. Liu, J. Cho, Analysis of User Web Traffic with a Focus on Search Activities, in: Proc. International Workshop on the Web and Databases (WebDB), 2005, pp. 103–108.

[18] C. Neasbitt, R. Perdisci, K. Li, T. Nelms, ClickMiner: Towards Forensic Reconstruction of User-Browser Interactions from Network Traces, in: Proc. CCS, ACM, New York, NY, USA, 2014. doi:10.1145/2660267.2660268.

[19] R. Perdisci, W. Lee, N. Feamster, Behavioral Clustering of HTTP-based Malware and Signature Generation Using Malicious Network Traces, in: Proc. USENIX NSDI, Berkeley, CA, USA, 2010.

[20] R. Perdisci, D. Ariu, G. Giacinto, Scalable fine-grained behavioral clustering of http-based malware, Computer Networks 57 (2) (2013) 487–500. doi:10.1016/j.comnet.2012.06.022.

[21] F. Tegeler, X. Fu, G. Vigna, C. Kruegel, BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection, in: Proc. CoNEXT, ACM, New York, NY, USA, 2012, pp. 349–360. doi:10.1145/2413176.2413217.

[22] H. Zhang, W. Banick, D. Yao, N. Ramakrishnan, User Intention-Based Traffic Dependence Analysis for Anomaly Detection, in: Proc. IEEE CS Security and Privacy Workshops (SPW), 2012, pp. 104–112. doi:10.1109/SPW.2012.15.

[23] P. Burghouwt, M. Spruit, H. Sips, Detection of Covert Botnet Command and Control Channels by Causal Analysis of Traffic Flows, in: Cyberspace Safety and Security, Springer International Publishing, 2013. doi:10.1007/978-3-319-03584-0_10.

[24] H. Shiravi, A. Shiravi, A. Ghorbani, A Survey of Visualization Systems for Network Security, IEEE Trans. on Visualization and Computer Graphics 18 (8) (2012) 1313–1329. doi:10.1109/TVCG.2011.144.

[25] R. Blue, C. Dunne, A. Fuchs, K. King, A. Schulman, Visualizing Real-Time Network Resource Usage, in: Visualization for Computer Security, Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-85933-8_12.

[26] R. Marty, AfterGlow, http://afterglow.sourceforge.net/, Last visited: 2014-06-30.

[27] NetWitness Visualize, http://visualize.netwitness.com/, Last visited: 2014-07-30.