Reconstructing Tabbed Browser Sessions Using Metadata Associations

Sriram V. Raghavan

Chapter 9

RECONSTRUCTING TABBED BROWSER SESSIONS USING METADATA ASSOCIATIONS

Sriram Raghavan and S.V. Raghavan

Abstract Internet browsers support multiple browser tabs, each browser tab capable of initiating and maintaining a separate web session, accessing multiple uniform resource identifiers (URIs) simultaneously. As a consequence, network traffic generated as part of a web request becomes indistinguishable across tabbed sessions. However, it is possible to find the specificity of attribution in the session-related context information recorded as metadata in log files (in servers and clients) and as network traffic related logs in routers and firewalls, along with their metadata. The forensic questions of "who," "what" and "how" are easily answered using the metadata-based approach presented in this chapter. The same questions can help systems administrators decide on monitoring and prevention strategies. Metadata, by definition, records context information related to a session; such metadata recordings transcend sources.

This chapter presents an algorithm for reconstructing multiple simultaneous browser sessions on browser applications with multi-threaded implementations. Two relationships, coherency and concurrency, are identified based on metadata associations across artifacts from browser history logs and network packets recorded during active browser sessions. These relationships are used to develop the algorithm that identifies the number of simultaneous browser sessions that are deployed and then reconstructs the sessions. Specially-designed experiments that leverage timing information alongside the browser and session contexts are used to demonstrate the processes for eliciting intelligence and separating and reconstructing tabbed browser sessions.

Keywords: Tabbed browser sessions, metadata association, session reconstruction

1. Introduction

The main function of a browser application is to present a web service by requesting it of a server and displaying it in a browser window. In practice, whenever a browser engages in a web session, it records information in its log files, usually meant for troubleshooting purposes. The browser and the associated logs work as follows: when a browser application is deployed, a process is created that interacts with the network socket (through one or more open ports) to access services from the network. The browser application logs the server responses and caches one or more of the resources received. In a generic browser, a parent browser process controls the overall operation and launches one or more rendering engines and loads plug-in modules as required [5]. The parent process is responsible for disk interactions, the user interface and the network. Every time a new browser window or tab is spawned, a new browser process is initiated to monitor the modules executing within its framework. Over a period of time, browser applications have evolved to create and maintain multiple simultaneous sessions [1, 6].

A single browser session may be defined as a sequence of browser requests and corresponding server responses that are received by the browser application; to start with, these are pivoted on a single browser tab. As the interactions progress, a user may optionally open more tabs and possibly more windows, but the manner of recording session information remains the same. However, the volume of information recorded to answer the forensic questions "who," "what" and "how" can grow significantly over time. The proposed approach for reconstructing browser sessions by associating session log information with a network trace converts the problem to a progressive and incremental one, thereby enabling the forensic or security context usage to be natural, easy and real time.

There are several important aspects related to reconstructing browser sessions and the relative browser tab sessions. Specifically, browser sessions are reconstructed to understand the number of simultaneous sessions operated by the browser. Additionally, when a user selects a new tab or clicks on a link from an existing tab to open a new browser session on another tab, the browser application has a unique way of opening the session. This relative positioning of browsers can help identify if a causal relationship existed; however, this task requires further analysis of the individual sessions and parsing the HTML pages to identify the hyperlinks involved in creating the session. It is possible to establish if two or more browser tabs placed close together connect to the same domain server and then, if necessary, identify the sequence in which the web pages were opened by studying the hyperlinks in each of the pages.

Such an investigation involves significant parsing effort and can become challenging when obfuscated code or malware is involved. However, this is outside the scope of this work; in fact, Neasbitt et al. [7] study the problem of differentiating user-clicked pages from auto-load components that are parts of an active browser rendering.

This research establishes the feasibility of isolating multiple simultaneous browser sessions on multi-threaded browser applications based on browser application logs and network traffic logs at the host end. In order to do so, a metadata-based approach is used to determine the contextual relationships between artifacts within and across the browser logs and network packets. The proposed algorithm based on the identified metadata relationships is applied to specially-designed experiments for a usage scenario involving a Firefox application with five tabbed sessions across two browser windows.

2. Related Work

Xie et al. [11] and Neasbitt et al. [7] have recognized the importance of recovering browser activities to aid forensic analyses of web traffic traces in investigations of user-browser interactions involved in security incidents, albeit from a purely user standpoint. In doing so, it is recognized that most browser applications implement a recovery feature that enables the browser to restore a session when an application crashes or does not undergo a proper termination procedure [6]. The recovery features ensure continuity when the context breaks down or drastically changes for operational reasons. While these appear to be similar, they vary in their details; both have a definite need, but different goals. The purpose of this research is not to reconstruct sessions when browser session crashes are involved; rather, the intention is to reconstruct browser sessions regardless of the nature in which the browser applications were terminated. This is particularly necessary when the actions of a suspected perpetrator must be tracked sequentially to determine the why more than the how during an investigation. The ability to reconstruct user sessions independent of the underlying browser process and application execution contexts is emerging as a requirement when applications are subject to compromise. Such as approach is discussed in this chapter.

In order to reconstruct a communications session from network traffic, network packets can be analyzed to determine the parties engaged in the communications, time of communications and type of information exchanged; such analysis enables the recorded information to be organized sequentially into logical sessions. However, when browsers use multiple tabbed sessions, building such logical sequences for each ses-

sion, that too individually, is non-trivial [2]. In fact, browser logs do not contain sufficient information to identify the number of parallel sessions deployed [8]. It is, therefore, necessary to associate artifacts recorded across all independent sources and correlate the events to reconstruct the browser sessions.

The goal of this research is multi-fold and there are several important contributions. First, when reconstructing browser sessions, each request is mapped to its corresponding response. Further, when a web page has animations or active content, it often tends to initiate separate requests that are not part of the original page; these requests are initiated when the page is being rendered in the browser window. Such requests are identified and their responses, where applicable, are mapped. Second, not all responses are initiated based on their corresponding requests. In such cases, requests that are likely to initiate multiple responses are mapped and the responses are suitably chained to their corresponding origins to determine the communications streams to which they belong. Third, the procedure is automated as an algorithm designed to operate in high-bandwidth environments while enabling analyses of the computations and memory use during browser session reconstruction.

3. Multi-Threaded Browser Application Design

A browser subsystem [4] consists of several components, including a parent browser process, logging agent, one or more network sockets to maintain the browser sessions, plug-ins to decode interactive content and maintain encrypted sessions, and one or more rendering engines associated with the user interface. Popular browser applications include Internet Explorer (Microsoft), Chrome (Google), Safari (Apple), Firefox (Mozilla) and Opera (Opera). This research focuses on browser applications that have multi-threaded implementations.

In multi-threaded browser applications, there is exactly one parent browser process that manages the individual sessions using threads, regardless of whether they were initiated using browser (application) windows or browser tabs. Therefore, when tracing the path of these browser sessions to the respective browser windows or tabs, it is sufficient to establish a one-to-one relationship between the window or tab and the session that was supported. This is possible because there is exactly one process through which information passes from the network to the application.

The Mozilla Firefox browser, a multi-threaded browser application, contains a single parent process and spawns a thread for each suboperation. The Firefox browser launches a new process for the first

instance of a browser window and maintains individual web sessions as threads. Since browser windows and browser tabs are interchangeable, creating a second browser window (either independently or by separating a tabbed browser session) does not give rise to a new process, but it continues as a thread executing off the original parent process. In other words, regardless of the number of Firefox browser windows or tabs, there exists only one process with one or more threads maintaining the different web sessions. Since a single process manages multiple browser sessions, an active Firefox browser process consumes much more memory than other browsers that do not adopt this approach. Furthermore, from an implementation standpoint, the Firefox browser parent process executes purely as a 32-bit process regardless of the underlying hardware platform; and consequently, the addressable space for the entire browser application is limited to 4 GB, which is must be managed across multiple sessions.

Despite the existence of multiple browser threads that process simultaneous sessions, web requests and responses are interleaved when they are logged by the application; the logged events appear as a single session because the application does not log information related to the number of simultaneously active sessions. However, a systematic analysis based on the proposed methodology can break down this apparent singular session into the respective tabbed sessions as described below.

4. Mapping Browser Actions

A browser process localizes itself with respect to its network when it is activated (i.e., an active session is initiated by a user). This activity requires a series of message exchanges between the local host and the network. Typically, this involves a link-local host localization that is achieved at layer-2 of the network stack. Following the localization, the host engages in name resolution with one or more DNS servers listed in its network registry. This action is followed by a TCP engagement with the server itself.

The browser application can engage with one or more TCP sessions with the server. When a new server needs to be identified, it is preceded with the name resolution phase. The sessions can be TCP or secure TCP (i.e., HTTPS on server port 443). The number of sessions is dependent on the original server response and can vary depending on the context. When multiple TCP sessions are in play, a browser can maintain multiple sessions using multiple network ports on the browser host machine; each session is maintained until the browser host sends a FIN request to terminate the session. During a browser session, a user may make

additional web requests until the user terminates the browser window or tab.

4.1 Browser Sessions and Logging

At any time, an active browser session requires the browser host machine to identify its position in the network (using ARP request-responses) before requesting information about the server from where a service is to be requested (using DNS request-responses). The network packets transmitted and received during this period (ARP-DNS-TCP/UDP-HTTP) can be sensed and their network packet attributes can be used as parameters associated with the corresponding browser session.

Consider the sequence of network transactions that occur when a user attempts to download a resource from the Internet. Initially, the user makes a web request through a web browser, which initiates an ARP request to identify itself within its local network (or subnet), following which it makes a DNS request to the local web proxy to identify the IP address of the server where the resource resides. After the DNS response is received, the browser host machine initiates a TCP connection with the particular server and makes a request for the resource. This request may or may not include an HTTP session, where the web server responds to the browser host machine with one or more web pages through which the resource request can be made. After the request is made, the resource is transmitted from the server to the browser host machine as a file transfer action, which is, in essence, a sequence of TCP packets.

This behavior of the browser session along with network address resolutions and network-based communications leave adequate information about all the activities and the associated events. This information can be analyzed as required during reconstruction. The most significant aspect of browser session reconstruction is that neither the browser log file nor the network packets captured have any information about the specific browser window or browser tab responsible for generating a web session. For each browser event that is a web response from some server, the browser logs the URL corresponding to the request, the title of the page as defined on the server, the domain in which the server resides (e.g., `google.com`), the timestamp corresponding to the response in the UNIX time format and structure information specifying how the response is to be rendered by the browser. As a result, a complete and rich web session in a browser window is represented in the form of a textual log entry. Normally, logs do not record the numbers of active browser windows or tabs and, therefore, reverse tracking remains a challenge.

Xie et al. [11] have proposed a method for reconstructing web browser activity by pruning a referrer-based graph; specifically, the method reconstructs web surfing activities from traffic traces by analyzing the referrer headers. Neasbitt et al. [7] have implemented the ClickMiner system for reconstructing user-browser interactions from network traces. ClickMiner uses referrer-click inference to prune a user browser activity graph based on referrer re-redirections to ascertain the points where a user actively participated in generating new web requests. Through a user study, Neasbitt et al. demonstrated that ClickMiner can correctly reconstruct up to ninety percent of user-browser interactions.

Raghavan and Raghavan [10] have presented an approach for identifying the source of a downloaded resource based on relationships established using browser logs, network packets and file metadata. They demonstrated the use of metadata-based associations to determine relationships between different sources of digital evidence (e.g., user filesystem, browser logs and temporary Internet files) to determine the origins of digital image files downloaded from the Internet. The metadata in a file is used to track alternate copies of the file and log events that created or affected the file during a user session. Using metadata associations, they determined file-to-file, file-to-log-event and log-event-to-log-event relationships, which were then traced to the source URL of the downloaded file. This research extends the work presented in [10] to identify the causal relationships between event sources. In particular, network parameters and browser history logs are used to distinguish as well as identify concurrent events across sessions and coherent events belonging to a single session while handling multiple simultaneous browser sessions of a browser application.

4.2 Tracking a Browser Session

When a browser interprets a web response and displays the corresponding HTML page, many aspects of the response can be logged by the browser. Normally, some of the entries that can be recorded in the browser history log are the page rendered (i.e., URL), domain corresponding to the URL, date and time when the web response was received, name of the referral server, number of visits in a given duration and the user under whose account the access was logged. Browser windows and tabs are interchangeable in most browser applications and the window or tab that generates a specific web response is not recorded or logged. As a consequence, the entries in the browser history log file appear in a sequential manner. Moreover, the web request responsible for the response is transparent to the utility that logs browser history in-

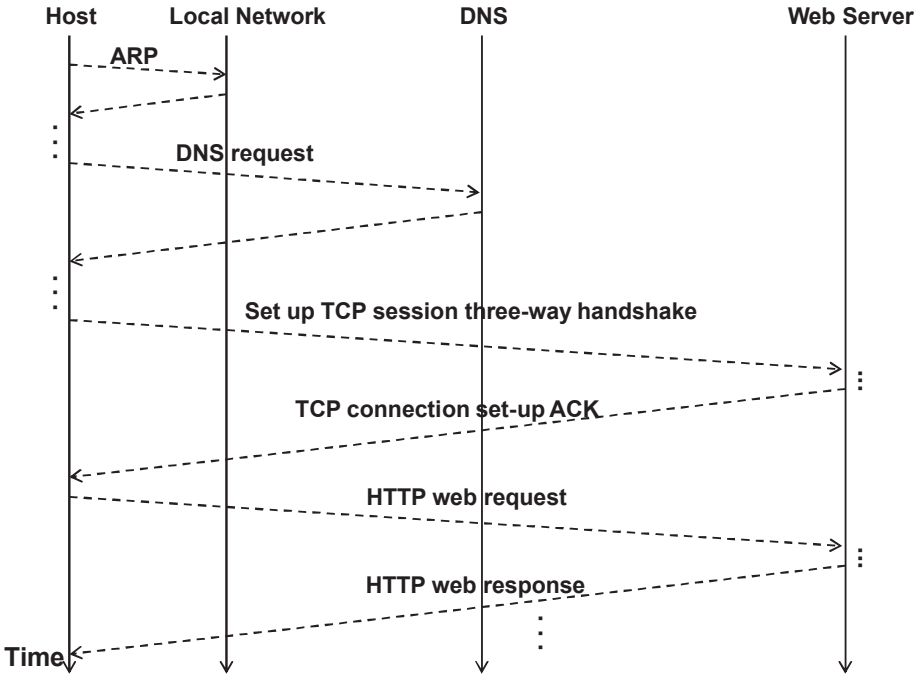

Figure 1. Space-time diagram corresponding to a web request.

formation. The missing relationship information must be obtained from the network packet log.

When a packet capture utility captures network packets entering and leaving a host machine, it uses filters to recognize and log network packets according to the protocol used in the communications. A simple web request leaving a host machine requires the machine to send ARP packets to identify the gateway in a given network and then request address resolution of the server with which the browser is attempting to communicate. After the address resolution yields an IP address, the browser attempts to set up a TCP channel with the server, following which the request is sent. Figure 1 displays this sequence as a space-time (S-T) diagram.

Note that a large number of packets enter and leave the host machine, but only the browser process makes the web request while the network stack on the host handles the rest internally. As a result, the HTTP request contains insufficient information to enable a trace back to a specific browser window or browser tab. Therefore, it is essential that any proposed method relies on a sequence of transactions that exhibit the characteristics shown in Figure 1 instead of a single HTTP request.

Each browser response in the browser history log is associated with a sequence of network packet transactions that correspond to the web request made by the browser on the host.

While the browser application tracks web responses that are recorded sequentially, network packets are needed to decipher web requests. Additionally, the packet capture must be timed so that it coincides with the browser sessions in question. This requires the capture utility to be positioned to have complete visibility of the sequence of network packets exchanged in the same sequence as shown in the space-time diagram in Figure 1. Such a task can be achieved in any organizational LAN by triggering a packet dump for the machine host where a browser application is launched and filtering for network packets initiated by the host in question.

5. Eliciting Session-Based Relationships

The browsing application history is stored on the host machine when browsing the web. It includes information entered into forms, passwords and sites that were visited. The browser history log records the requests sequentially; in the presence of multiple tabs, subsequent requests across tabs are interleaved as the browser continues to generate additional web requests. In such a scenario, network packets are collocated in time, but belong to different browser sessions. It is interesting to note that the browser tabs (browser session) may generate web traffic and network packets to different servers, thereby establishing distinct streams of web responses and network packet flows. Associating a browser log event with its corresponding sequence of network packets requires the careful identification of the relevant characteristics of the browser log entry that can aid this activity. Figure 2 shows the generic layout of a browser history log entry [8] along with the metadata of interest.

Network packets have many useful attributes (or metadata) that may be used to associate the packets with a particular network session. The information includes the source and destination IP addresses, protocol, timestamp corresponding to when a packet was seen leaving or entering the browser host machine, host browser (TCP) process port and server port numbers, and session sequence number. Figure 3 shows the generic layout of a network packet [3] with the relevant metadata highlighted.

The proposed approach identifies specific network packet sessions in accordance with the space-time sequence shown in Figure 1 and constructs a high-level transaction that results from the web request generated by the browser process. As discussed in the previous section, these have one-to-one correspondence with the browser history log entries.

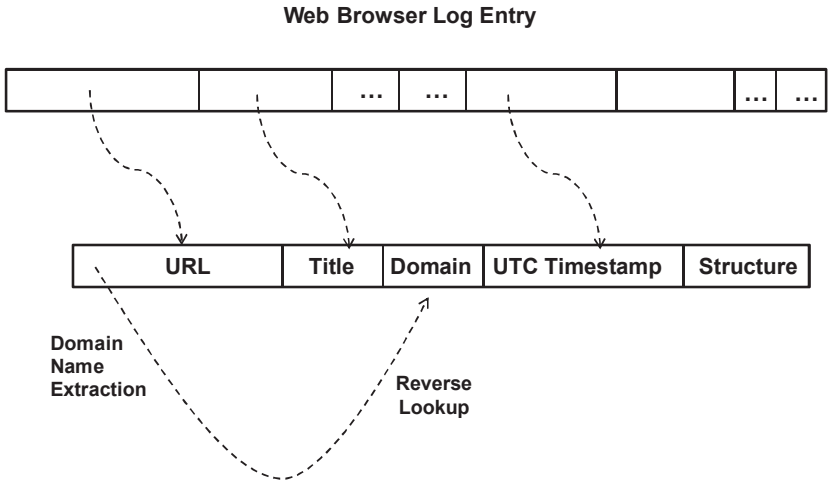

Figure 2. Browser history log entry with relevant metadata.

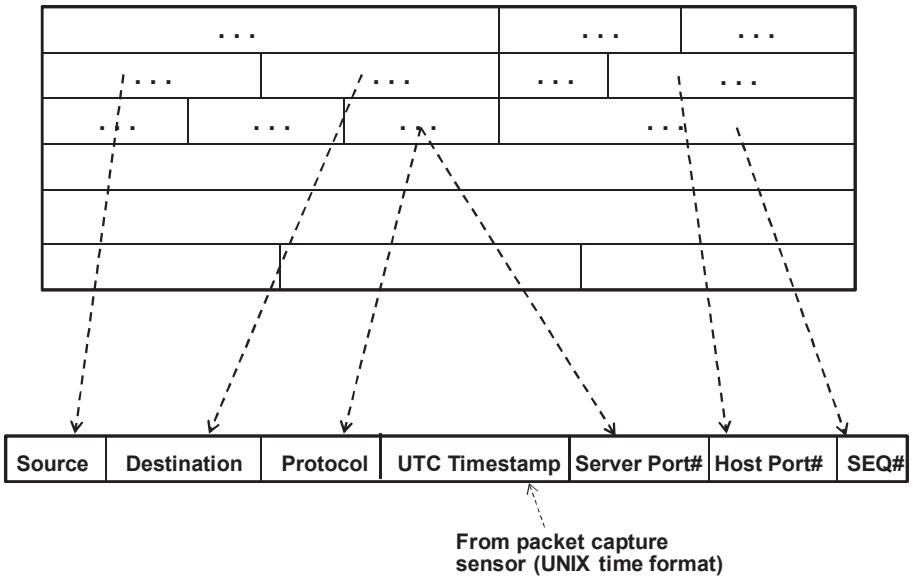

Figure 3. Generic layout of a network packet with relevant metadata.

The distinction across simultaneous browser sessions (using multiple windows or tabbed sessions) is not apparent when a browser log or network packet trace are seen in isolation. In contrast, the browser logs and network packets together provide a session context that the proposed approach leverages to elicit the relationship between the two

information sources. This relationship is extracted using knowledge of the sequence of actions that govern a browser-based interaction (space-time diagram) and the timing information associated with the network packets exchanged between the host and the respective web servers with which the browser application on the host interacts. This is discussed further in the following subsections.

5.1 Modeling Browser Sessions

A browser session is modeled by selecting a browser that satisfies the specifications of this research (e.g., Mozilla Firefox browser) and a basic static homepage (e.g., `google.com`). This ensures that the homepage consumes minimum additional memory and time as it loads and also allows a focus on the dynamics of the browser rather than on the web page. A list of web pages is prepared that contain a mixture of static pages as well as dynamic pages with animations and plug-ins. In sequence, additional tabs are created in an open browser window with each of these pages opened one at a time; each time, the increase in memory consumption, network ports used, number of additional network connections established and the memory locations used by the multi-threaded browser process are noted. This provides an estimate of how much additional memory is consumed when a page is opened as a second tab, where the first tab is always set to the homepage. This exercise is then repeated on a browser window for each of the web sites. In each case, the additional tab or window opened that directs to the homepage is used as a control.

In the next step, keeping two fixed tabs, the exercise outlined above is repeated for each of the websites in the list and records are made of the memory consumed, network ports used, number of additional network connections established and memory locations used by the browser process. This is repeated for the browser windows in the same manner. Iterations are performed from two tabs to eight tabs and observations are made about how the browser behaves along with the parameters listed above. The case is also repeated for browser windows starting with two and increasing the number of windows in steps to eight and noting the parameters for each case. This process provides a representation of a browser session.

In the case of a single browser session, network packets are causally related with the respective browser events that, in turn, are structurally tied to the session – this can include the use of a specific process $\mathbf{p}$ and network ports n_1, n_2, n_3 to send and receive the network packets corresponding to the browser web request and its corresponding server

response. Naturally, all the communications sent and received during the session are maintained through the network ports that bind them structurally to the browser session tab.

5.2 Developing a Browser Session State Space

An active browser session is characterized by a web request q, a browser host forward network session np_{fwd}^{host}, server response r and a browser host return network session np_{ret}^{host}. The network sessions np_{fwd}^{host} and np_{ret}^{host} are maintained by a browser process $\mathbf{p}$ on network port n. Thus, a simple browser session s with a single web request followed by a single server response is represented as:

$$s = (q, np_{fwd}^{host}, r, np_{ret}^{host}, \mathbf{p}, n) \tag{1}$$

When the browser maintains the session over a period of time T, it can make multiple web requests q_1, q_2, q_3 and receive an equal number of responses r_1, r_2, r_3 for a specific browser process $\mathbf{p}$ on network port n. The web requests q_1, q_2, q_3 are grouped into a request set Q and the server responses r_1, r_2, r_3 into a response set R. As shown in Equation (1), each web request from the host machine is associated with a unique network port number and timestamp. When a server response is observed for the host on the same network port, the association is established on a first-in-first-out basis. Server responses observed on other ports meant for the same destination are identified as referred responses launched by the browser during page rendering. In this manner, user requests are separated from referred requests and a one-to-one correspondence is established between the elements in set Q and the elements in set R.

The sequence of network packets transmitted from the browser host machine is grouped in set NP_{fwd}^{host} while the sequence of network packets received at the browser host machine from the server is grouped in set NP_{ret}^{host}. Timing information associated with session information is quite critical in the reconstruction. This requires the association of time with the progress of a session that can be utilized during the reconstruction. The proposed approach does not represent time explicitly; however, readers should note that every request and response observed in the network is implicitly associated with a timestamp. As a consequence, both NP_{fwd}^{host} and NP_{ret}^{host} have elements (individual requests and responses seen in the network) that are mapped one-to-one on the timeline. Therefore the browser session S maintained by browser process $\mathbf{p}$ on network port n over a time period T is given by:

$$S = (Q, NP_{fwd}^{host}, R, NP_{ret}^{host}, \mathbf{p}, n) \tag{2}$$

Using this representation of the state space of a single browser session, two relationships are elicited to define a coherent session and distinguish it from a concurrent session. A coherent session is one where the tagged activities correspond to a single browser session. A concurrent session is one where the tagged activities co-occur in time and belong to distinct browser sessions without any further relationships. These relationships are defined by matching metadata values. A metadata value match is an exact match or a threshold-based match using a predetermined threshold δ to accommodate network delays. The two relationships are used to group related artifacts from the recorded evidence and reconstruct parallel browser sessions across multiple browser tabs.

5.3 Coherent Event Relationship

When a metadata match occurs across two artifacts a_1 and a_2 in a browser session S, where a_1 and a_2 belong to the events from the browser log or from observed network traffic originating from the same server (domain), a coherent event relationship exists when the two artifacts a_1 and a_2 belong to a single browser session. The coherent event relationship is expressed as $a_1 R_{coh} a_2$. By definition, the relationship R_{coh} is reflexive and associative:

- $a_1 R_{coh} a_2 \Leftrightarrow a_2 R_{coh} a_1$
- $(a_1 R_{coh} a_2) \cap (a_2 R_{coh} a_3) \Rightarrow (a_1 R_{coh} a_3)$

When all the artifacts $a_1, a_2, a_3, \ldots, a_m$ exhibit identical associations with each other, the relationship is expressed as $R_{coh}(a_1, a_2, a_3, \ldots, a_m)$.

5.4 Concurrent Event Relationship

When a metadata match occurs across two artifacts a_1 and a_2 in a subset of the state space (R, NP_{ret}^{host}) in a given time interval T, where a_1 and a_2 belong to the events from a browser log or from the observed network traffic packets, a concurrent event relationship exists when the two artifacts a_1 and a_2 share concurrency in time but belong to different browser sessions. The concurrent event relationship is expressed as $a_1 R_{ccn} a_2$. By definition, the relationship R_{ccn} is reflexive but not associative:

- $a_1 R_{ccn} a_2 \Leftrightarrow a_2 R_{ccn} a_1$
- $(a_1 R_{ccn} a_2) \cap (a_2 R_{ccn} a_3) \not\Rightarrow (a_1 R_{ccn} a_3)$

When all the artifacts $a_1, a_2, a_3, \ldots, a_n$ exhibit identical associations with each other, the relationship is expressed as $R_{ccn}(a_1, a_2, a_3, \ldots, a_m)$.

Algorithm 1 : Rachna – Browser session reconstruction algorithm.

Input: Browser request list $Q = \{q_1, q_2, q_3, ...\}$, server responses $R = \{r_1, r_2, r_3, ...\}$, observed network traffic at the browser host NP_{fwd}^{host} and NP_{ret}^{host}, browser process $\mathbf{p}$ and network ports $\{n_1, n_2, n_3, ...\}$

Output: Simultaneous browser sessions for each corresponding (Q, R)

num-sessions $\leftarrow 0$
for all num-sessions = ||largest set $R_{ccn}(...)$|| **do**
    if server response $R \neq \emptyset$ **then**
        num-sessions $\leftarrow$ num-sessions + 1
    end if
    for all server responses $r \in R$ **do**
        $l \leftarrow$ list of referenced resources found in the response
        for all resource items $\in l$ **do**
            Map the resource in l to TCP sessions from NP_{fwd}^{host}, NP_{ret}^{host}
        end for
        Create the state $(q, np_{fwd}^{host}, r, np_{ret}^{host}, \mathbf{p}, n)$ for server response $r \in R$ for each unique network port n
    end for
    num-sessions $\leftarrow$ num-sessions + 1
    Group all requests Q corresponding to responses in l for a single session to derive the session state $(Q, NP_{fwd}^{host}, R, NP_{ret}^{host}, (p), n)$ for browser process $\mathbf{p}$ for each unique network port n
    Stagger the session states so formed and order them chronologically with respect to the web requests in Q for each session state
end for
Display num-sessions as the number of simultaneous browser sessions
for all distinct sessions **do**
    Display $(Q, NP_{fwd}^{host}, R, NP_{ret}^{host}, \mathbf{p}, n)$
end for

When this condition holds, it can be interpreted as evidence of at least m distinct browser sessions because any two artifacts, taken two at a time, exhibit the concurrence relationship. With regard to ordering and prioritizing relationships, a coherency relationship supersedes a concurrency relationship where applicable. Consequently, if two artifacts a_1 and a_2 exhibit a concurrency relationship as well as a coherency relationship, then coherency is given priority and concurrency is dropped. This condition accounts for artifacts that belong to the same session, but may be recorded in parallel TCP sessions of a browser.

Algorithm 1 formalizes the process of reconstructing browser sessions. The algorithm is named Rachna, which means "to form" or "to construct" in Sanskrit.

Figure 4 shows the reconstruction of multiple browser sessions using network packets and the corresponding browser history log entries. The browser history log events show the web responses as presented by a

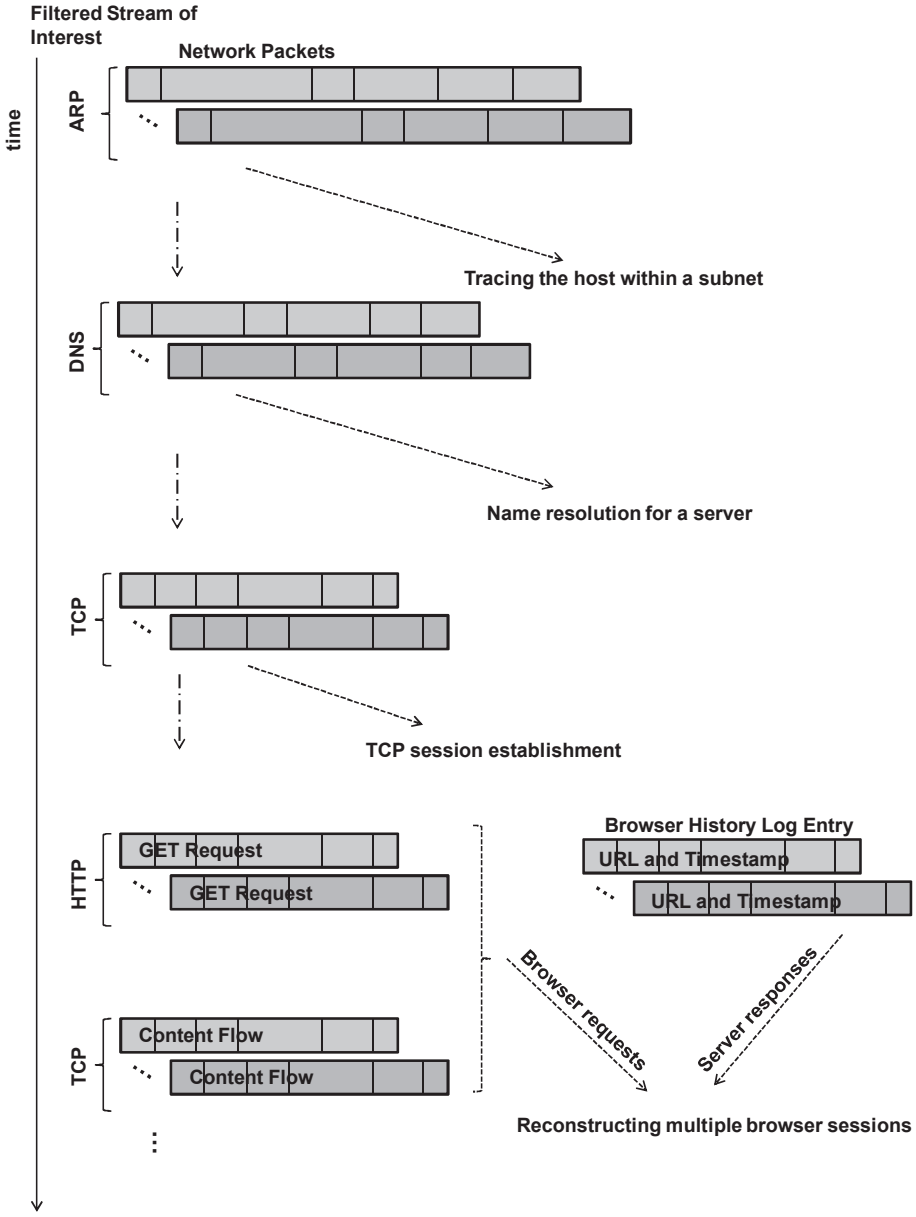

Figure 4. Reconstruction of multiple browser sessions.

browser window or tab. However, the request that led to a response being generated is implicit. Naturally, during the reconstruction it becomes necessary to trace its origin. Such a request is derived by parsing the

server response to determine the list of resources, which are referenced to identify their respective sources.

For each resource, the network sessions initiated by a host are correlated and the request-response sequences are grouped to create the tuple (q, np^{host}_{fwd}, r, np^{host}_{ret}, $\mathbf{p}$, n) for the browser process $\mathbf{p}$. Having identified such sequences, the number of simultaneous sessions deployed is updated.

The network packets in Figure 4 that are colored differently have a concurrency relationship. When network packets demonstrate a coherency relationship with a particular server response in set R, they are grouped together to indicate a coherent session. Such groupings are performed to reconstruct each coherent session.

Artifacts that do not exhibit coherency relationships have concurrency relationships because they do not share a session context beyond the obvious. Such artifacts are represented by $R_{ccn}()$. By definition, each pair of artifacts is concurrent and not coherent – thus, for each pair, there must exist two sessions that do not share a context. Hence, the cardinality of the largest set defined by $R_{ccn}()$ is the number of distinct browser sessions for a browser application and browser process $\mathbf{p}$.

6. Identifying Browser Artifact Relationships

An experiment was performed to identify the relationships between browser artifacts using metadata associations.

A user was asked to launch a Mozilla Firefox browser (with developer mode and network request tracker enabled) and conduct a browsing session. The user opened the browser application with three tabs on the first browser window and then opened a separate window with two additional tabs. The default homepage was set to `google.com`.

In the first window, the user connected to `securecyberspace.org` and the online TV news guide `sidereel.com`. In the second window, the user logged into his personal Yahoo! email account and the second tab was connected to `youtube.com`. Next, the user modified the browser tab connected to Yahoo! email to become a separate window. In each of the two browser tabs connected to `sidereel.com` and `youtube.com`, the user executed a search and selected an item from the search list to be played. The entire session was captured using Wireshark over a ten-minute period. The relevant browser history sessions were also recorded. The network packets and browser logs were then analyzed. Table 1 summarizes the dataset collected in the experiment. Since the entire session was closely monitored and tracked, Wireshark was used to

Table 1. Experimental dataset.

Evidence	Number
Browser history	32
GET resource requests	1,489
Network packets	52,692

extract the SSL session keys and decipher the encrypted sessions after obtaining permission from the user.

The AssocGEN analysis engine [9] was adapted to identify the coherency and concurrency relationships based on the metadata determined from the browser log entries and network packets captured during the active network sessions. The browser history log entries and the individual network packets associated via their respective parameters identified in Figures 2 and 3, respectively, were grouped together depending on the relationships exhibited. Specifically, the simultaneous tabbed browser sessions that exhibited coherent event relationships on artifacts belonging to a single tabbed session exhibited both causality and session dependence (browser process to port binding) across browser log entries and network packets. This was used to separate the different simultaneous sessions.

The number of sessions was initially set to zero and, because the server response was non-null, the number of sessions was incremented to one. The original HTML responses for each session were parsed to obtain a total of 1,489 resource requests spread across five browser sessions and more than 52,692 network packets captured over the entire browsing duration. To eliminate requests generated from the browser during the rendering process, metrics provided by the developer mode of the Firefox browser were used to isolate HTML content from the remaining elements such as JavaScript, JPG, CSS and XML.

The origin server was identified for every reference resource that was downloaded. Each time a response from a new server was identified, the number of sessions was incremented. After the origin server for each resource was identified, coherency relationships between the corresponding server responses and network packets were discovered. The coherent artifacts (server response r, resource list l, network traffic np_{fwd}^{host}, network traffic np_{ret}^{host}) were grouped under a single browser session. As described in the algorithm, each referrer connection initiated by the browser during rendering was conducted on a network port other than the one where the main HTML page was being rendered. This enabled the main elements of the page and the add-on elements to be distinguished. The

distinction enabled the linking and chaining of server responses from the streaming server (`youtube.com`), where a single request can contain a response that initiates a new request.

While establishing the coherency between the artifacts from a single browser session, resource lists across sessions were compared with the aggregated network traffic at the browser host machine. This revealed concurrency relationships between the resources and network packets. Next, the largest set of artifacts for which concurrency relationships could be established was determined ($\|R_{ccn}\|$). The number of sessions was updated for each new element discovered in this set; in the experiment, the number of sessions was five.

7. Results and Discussion

After the artifacts were grouped according to the R_{coh} and R_{ccn} relationships, the network traffic that serviced each web request sent by the browser was aggregated. Thirteen distinct web requests were generated by user browser actions during the period of observation. The host was isolated using the network traffic; this revealed that the browser had a total of five tabbed browser sessions.

Two static page connections from the first browser window were identified (`google.com` and `securecyberspace.org`). These connections did not generate repeated reconnections and it was determined that the sessions were associated with a single browser window. This was determined based on the memory locations allocated to the browser window (first window), which contained the session context in memory for browser process **p**. However, the user subsequently requested a page named "Practicing Security" on the page connected to `securecyberspace.org`. No further activity was observed on the pages.

The tabbed sessions connected to the two online media servers generated the most rendered requests: 189 resource requests for `sidereel.com` and 158 resource requests for `youtube.com`. The pop-up and dynamic prompts were not included in the count. During the analysis, these request elements did not provide a precise source of origin; this contributed to the ambiguity. The connection with the email server also had a large number of requests, although they were restricted to the server homepage. After the user requests progressed to the mail login pages, the number of requests were reduced drastically. It should be noted that this was primarily due to the advertisements provided on the website that contained referred elements from the parent server (i.e., `au.yahoo.com`), which generated new connections on behalf of the server. Further analysis of the browser cache and stored information on

Table 2. Reconstructed browser sessions.

Tabbed Session ID	Timestamp	URL
1	13-08-2015 PM 02:52:15	www.google.com
2	13-08-2015 PM 02:52:37	www.securecyberspace.org
3	13-08-2015 PM 02:53:12	www.sidereel.com
2	13-08-2015 PM 02:53:28	http://www.securecyberspace.org/practicing_security
4	13-08-2015 PM 02:55:32	https://www.youtube.com
5	13-08-2015 PM 02:55:57	https://au.yahoo.com/?p=us
3	13-08-2015 PM 02:56:46	www.sidereel.com/_television/search?utf8=%E2%9C%93&q=thewestwing
4	13-08-2015 PM 02:57:21	https://www.youtube.com/results?search_query=arrow+season+2
3	13-08-2015 PM 02:57:58	http://www.sidereel.com/The_West_Wing
5	13-08-2015 PM 02:59:35	https://login.yahoo.com/config/mail?&.src=ym&.intl=au
5	13-08-2015 PM 02:59:46	https://edit.yahoo.com/config/change_pw?.done=https%3A%2F%2Fmail.yahoo.com&.src=ym&.intl=au&.spreg=4&.scrumb=z9e3gjYzlQb&.lang=en-AU&.asdk_embedded=&.appsrc=&.appsrcv=&.srcv=&.smc=&sts=1441272728&sig=2c1bb1d2
4	13-08-2015 PM 02:58:36	https://www.youtube.com/watch?v=wYn0I9gtoKw
5	13-08-2015 PM 03:00:17	https://edit.yahoo.com/config/change_pw?.scrumb=z9e3gjYzlQb&.done=https%3A%2F%2Fmail.yahoo.com&.src=ym&.st=4
5	13-08-2015 PM 03:01:42	https://au-mg5.mail.yahoo.com/neo/launch?.rand=4vfcsaqi45krv

the host machine helped identify the user's stored email login credentials (recovering user details and identifying the account accessed by the user are added benefits of brower sesssion reconstruction). The email server and youtube.com repeatedly refreshed their page contents identified in page elements such as JavaScript, JPG and XML, and some ActiveX content generated new host-server connections on new network ports.

7.1 Results

Table 2 presents the reconstructed browser sessions (server responses R). The results were validated by repeating the activity on the same

browser providing the URL as identified by the browser history log for each tabbed session and comparing the generated records. The analysis engine was used to first process the browser history logs and load the log records and their parsed metadata into the engine repository. After the browser logs were traversed completely, the network packets obtained during the capture were traversed. Following this, a procedure call was used to generate all object relationships based on associations identified in their metadata across the browser history log entries, TCP connections, browser process information and network port information from memory, and network packets in the packet capture.

After the associations were generated and stored in the repository, the syntactic relationships between artifacts of the same type (i.e., among network packets and browser history logs) and the semantic relationships across types were elicited to discover the origins of the web sessions. Artifacts that belonged to the same application were determined to have coherent event relationships R_{coh}. This is typically true for all records from a browser history log or between network packets.

The concurrent event relationships R_{ccn} were determined to exist between artifacts that occurred at the same time but contained different session contexts, and belonged to distinct browser sessions. This is true of browser history records captured across tabs running simultaneous sessions or between network packets that service parallel sessions across different browser tabs. Based on the reconstruction, the sessions were replayed on a Firefox browser to corroborate the evidence. Figure 5 shows a snapshot of the replayed Firefox browser sessions consisting of three browser windows and five tabbed browser sessions. Note that, while the algorithm could distinguish the sessions carried out across two browser windows and over five tabbed sessions, the algorithm did not discern the change when the two browser windows were expanded to three windows during the session. It is believed that the browser process **p** maintains the same memory locations and session ID to service this session, but the special identifiers that separate a tabbed session as a new browser window remain to be identified. Current research is exploring this aspect of browser behavior.

7.2 Discussion

The proposed method for distinguishing multiple simultaneous sessions leverages artifact relationships derived from metadata-based associations. Coherency relationships are employed to establish connectedness between artifacts belonging to a single session and concurrency relationships are used to distinguish between artifacts that share the

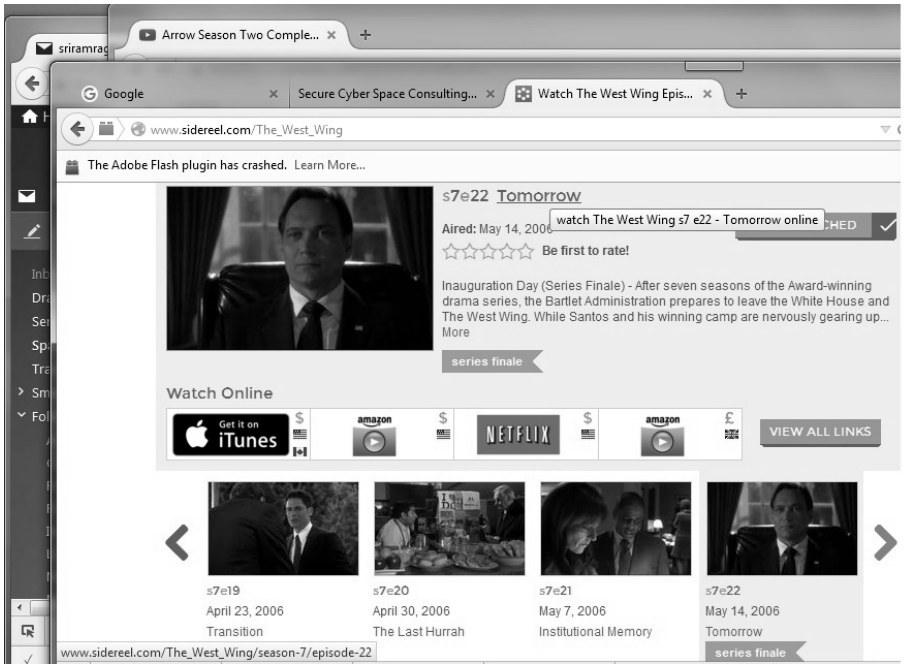

Figure 5. Reconstructed Firefox browser sessions.

time of occurrence, but not the session context. However, some special cases must be considered due to the assumption that multiple browser sessions may exist at the point of initiation.

It is also necessary to consider cases where new tabs are created along the way and to detect them effectively using the Rachna algorithm, especially when the new session communicates with the same server as the immediately-previous tab. In this case, while the resources from the server can be mapped after parsing the original server response as described in the algorithm, the newly-deployed tab may share resources with the original session and this may not warrant a new network session – this could create a miss when the resources list is mapped against the network traffic. This can be identified by tracking the network ports on the host to determine when a new connection is requested from a server that has an existing connection with browser process **p**. There is some scope for refining the detection method in such an approach.

A browser application can host one or more browser sessions, which are hosted on one or more browser windows. The transactions conducted across such sessions are recorded in the browser logs. The information processed by the browser application is volatile. Between the parent

process and the threads maintaining multiple browser sessions, there are two-way communications that ensure that each thread receives its unique commands.

When the parent process needs to create and maintain a network session, it makes a request to the network stack and this communication is unidirectional – in the sense that the network stack does not track the origin of the request. All the completed network services are generically returned through the network stack to host memory (shared memory implementation), from where the process responsible for making the call reads the information for further processing. It was also observed that the Rachna algorithm could not deal with pop-up windows and dynamic prompts with page timeouts. This is because the algorithm cannot associate the precise origin of a pop-up on a page with a request originating from the host machine.

While the approach may appear to be efficient in terms of operating the application and not losing information when sessions may need to be sandboxed, the lack of explicit two-way attribution remains a challenge with regard to efficient and timely reconstructions performed when investigating security incidents. Providing attribution by ensuring two-way information recording can go a long way in developing attack-resilient browsers in the future.

8. Conclusions

This research has demonstrated the feasibility of isolating multiple simultaneous browser sessions on multi-threaded browser applications based on browser application logs and network traffic logs on a browser host machine. Metadata-based associations were leveraged to identify and reconstruct all tabbed sessions that are part of a typical browser interaction. Coherency and concurrency relationships between artifacts derived from browser history logs and network traffic were used to identify the number of simultaneous sessions. The Rachna algorithm was developed to identify these relationships, discover the number of simultaneous browser sessions deployed and reconstruct the sessions. The effectiveness of the approach was demonstrated by conducting an analysis of a five-tabbed Mozilla Firefox browser session reconstructed using browser history logs and network packets. Finally, it was shown that the cardinality of the largest set of concurrent artifacts from a collection can be used to identify the number of simultaneous sessions by tracking the main page elements and distinguishing them from the rendered elements and the active elements that request reconnections from the same server.

Many current browsers provide an "incognito" or "silent" option to prevent browser activities from being recorded in browser application logs. This can have a significant impact on the reconstruction method presented in this chapter and may impede the identification of coherency and concurrency relationships. However, the incognito mode simply cuts off the logging module while the actual information continues to reside with the application and is usually held by the parent process. Future research will extend the approach to deal with such scenarios. Also, research will explore the minimum information needed to reconstruct browser sessions with acceptable accuracy. Many active pages, especially those associated with online media, contain pop-ups and dynamic prompts; current research is attempting to identify the distinguishing features of a pop-up on a web response and to incorporate these features in the Rachna algorithm. The knowledge gained from these efforts may lead to fundamental changes in the specification of future Internet-related active devices and applications.

References

[1] Chromium Projects, Multi-Process Architecture (`www.chromium.org/developers/design-documents/multi-process-architecture`), 2016.

[2] M. Cohen, PyFlag – An advanced network forensic framework, *Digital Investigation*, vol. 5(S), pp. S112–S120, 2008.

[3] G. Combs, Wireshark (`www.wireshark.org/about.html`), 2016.

[4] A. Grosskurth and M. Godfrey, A reference architecture for web browsers, *Proceedings of the Twenty-First IEEE International Conference on Software Maintenance*, pp. 661–664, 2005.

[5] N. Lwin, Agent based web browser, *Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems*, pp. 106–110, 2009.

[6] Mozilla, Mozilla Browser Architecture, Mountain, View, California, 2014.

[7] C. Neasbitt, R. Perdisci and K. Li, ClickMiner: Towards forensic reconstruction of user-browser interactions from network traces, *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1244–1255, 2014.

[8] J. Oh, S. Lee and S. Lee, Advanced evidence collection and analysis of web browser activity, *Digital Investigation*, vol. 8(S), pp. S62–S70, 2011.

[9] S. Raghavan and S. Raghavan, AssocGEN: Engine for analyzing metadata-based associations in digital evidence, *Proceedings of the Eighth International Workshop on Systematic Approaches to Digital Forensic Engineering*, 2013.

[10] S. Raghavan and S. Raghavan, Determining the origin of downloaded files using metadata associations, *Journal of Communications*, vol. 8(12), pp. 902–910, 2013.

[11] G. Xie, M. Iliofotou, T. Karagiannis, M. Faloutsos and Y. Jin, ReSurf: Reconstructing web-surfing activity from network traffic, *Proceedings of the IFIP Networking Conference*, 2013.