Reconstructing Tabbed Browser Sessions Using Metadata Associations for Multi-Threaded Browser Implementation

Sriram Raghavan  
Department of Computing & Information Systems  
University of Melbourne, Parkville VIC 3010  
Email: sriram.raghavan@unimelb.edu.au

S V Raghavan  
Department of Computer Science & Engineering  
Indian Institute of Technology Madras, Chennai, TN India  
Email: sv.raghavan@gov.in

Abstract—Today, Internet browsers support multiple browser tabs, each browser tab capable of initiating & maintaining a separate web session, accessing multiple URIs simultaneously. As a consequence, the network traffic generated as part of a web request becomes indistinguishable across tabbed sessions. But one can 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 question of “who”, “what”, and “how” are easily answered using the metadata based approach in this paper. From security viewpoint, the same questions help the administrator decide on “monitoring” and “prevention” strategies. Metadata, by definition, records context information related to a session. Such metadata recordings transcend sources.

In this paper, we propose an algorithm to reconstruct multiple simultaneous browser sessions on browser applications having a multi-threaded implementation. We define two relationships called coherency and concurrency, identified based on metadata associations across the artifacts from browser history logs and network packets recorded during active browser sessions. We use these relationships to develop an algorithm called “Rachna” to identify number of simultaneous browser sessions that are deployed and reconstruct those sessions. We demonstrate through specially designed experiments that use timing information alongside the browser and session context, the process to elicit intelligence and the process to separate out the tabbed sessions.

Keywords—Browser session, Browser tabbed sessions, Metadata association, Coherent event relationship, Browser Session reconstruction, Session context, Concurrent event relationship.

I. 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 the 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 that interacts with the network socket (through one or more open ports) to access services from the network is created. The browser application logs the server responses and caches one or more of the resources received. In the generic browser, a parent browser process controls the overall operation and is responsible for launching one or more renderer engines and load plugin 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 starts to monitor the modules executing within its framework. Over a period of time, browser applications have evolved and have the ability to create and maintain multiple (simultaneous) sessions [2, 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, pivoted on a single browser tab to start with. As the interaction progresses, user may optionally open more tabs and perhaps more windows too. However, the manner of recording session information continues to remain the same. While answering the questions “who”, “what, and “how”, the volume grows significantly over time. Our approach to reconstructing the browser sessions by associating session log information with the network trace converts the problem to a progressive and incremental one, thereby enabling the “forensic” or “security” context usage natural, easy, and real-time.

There are many important aspects to reconstructing the browser sessions and the relative browser tab sessions that are informative. Particularly, we reconstruct the browser sessions to understand how many simultaneous sessions were operated by the browser. Besides, 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 such a session. This relative positioning of the browsers will help identify if such a causal relationship existed and although this task requires further analysis of the
individual sessions and parsing the HTML pages to identify the hyperlinks involved in creating the session. If two or more browser tabs placed closed together connect to the same domain server, this can be established from our work and then further investigated, if necessary, to identify the sequence in which these web pages were opened by studying the hyperlinks in each of those pages. Such an investigation involves significant parsing efforts and when obfuscated code or malware is involved, it can become challenging. However, this is not in the scope of our work. We note that Neasbitt et al. [10] study this problem to differentiate user-clicked pages from auto-load components that are parts of active browser rendering.

In this paper, we establish the feasibility of isolating multiple simultaneous browser sessions on multi-threaded browser applications based on the browser application logs and the network traffic logs at the host end. In order to do so, we adopt an approach based on metadata and determine the contextual relationships between the artifacts within and across the browser logs and network packets. The algorithm based on the metadata-relationships identified that we develop in this work are applied to specially designed test-bench for a usage scenario with the Firefox browser consisting of five browser tabbed sessions across two browser windows in Section VI. In fact, it is this brevity and elegance that compels one to adopt progressive and incremental approach in the design. We review related research to motivate our work in the sequel.

II. RELATED WORK

Xie et al. [11] and Neasbitt et al. [10] have also recognized the importance of recovering browser activities to aid the forensic analysis of web traffic traces in the investigation of the user-browser interactions involved in a security incident, albeit from a purely user’s standpoint. In doing so, we recognize that most browser applications today implement what is known as a “recovery feature” that allows the browser to restore browser sessions when the application crashes or does not undergo proper termination procedure [6]. The recovery features available at present are for ensuring “continuity” when context breaks down or drastically changes for operational reasons. While they appear similar they do vary in detail. Of course, it is needless to add that both have a definite need but a different goal. Our purpose is not to reconstruct sessions when browser session crashes are involved; we intend to reconstruct browser sessions irrespective of the nature in which the browser application was terminated. This is particularly necessary when a suspected perpetrator’s actions must be sequentially tracked in order to determine the “why”’s more than the “how”’s during an investigation. The ability to reconstruct user sessions independent of the underlying browser process context and application execution context is emerging as a requirement when such applications are subject to compromise. We investigate such an approach in this paper.

For reconstructing a communication session from network traffic, the network packets can be analyzed to determine the parties engaged in communication, time of communication, and type of information exchanged; such analysis enables one to organize the recorded information in to logical sessions sequentially. But, when browsers use multiple tab sessions, building such logical sequences for each session, that too individually, becomes non-trivial [1]. In fact, browser logs do not contain sufficient information to identify the number of parallel sessions deployed [4]. It is therefore necessary to associate artifacts recorded across all independent sources and correlate the events, in order to reconstruct the browser sessions.

Our goal is multi-fold and there are several important contributions from this work. Firstly, when we reconstruct the browser sessions, we map each request with its corresponding response. Further, where web pages have animate or active content, they often tend to initiate separate requests which are not part of the original page but the requests are initiated when the page is rendered in the browser window. Such requests are identified and their responses, where applicable, are mapped. Secondly, not all responses are initiated based on its corresponding request. In such cases, we map those requests that are likely to initiate multiple responses and suitably “chain” these responses to their corresponding origins to determine the stream of communication they belong to. Thirdly and lastly, we automate the procedure so developed to operate on high-bandwidth environments while at the same time study the computation and memory performance of the algorithm during browser session reconstruction.

In our work, we use “metadata based associations” to identify and reconstruct all tabbed sessions that are part of a typical browser interaction. The rest of the paper is organized as follows: In Section III, we review the design of a browser application and identify the challenges in reconstructing browser sessions on multi-threaded browser applications. In Section IV, we map the different actions that occur within a host and over the network when a browser initiates one or more browser sessions. In Section V, we identify the information commonly available on browser logs and network packets and develop session based relationships to distinguish between events belonging to a single session and those that occur simultaneously across different sessions. Further, we develop an algorithm to identify these relationships from browser logs and captured network packets on the browser host. In Section VI, we describe the experimental evaluation of our method using a specially designed test-bench and the conduct of experiments using the Firefox browser. The main characteristic of our approach is the ability to perform with small size data. In Section VII, we discuss the results of our experiments and provide a discussion on the implications of this work. In Section VIII, we summarize our work and indicate possible future work.

III. GENERIC DESIGN OF A MULTI-THREADED BROWSER APPLICATION

The browser subsystem [3] consists of several components including a parent browser process, a logging agent, one or more network sockets to maintain the browser sessions, plugins to decode interactive content as well as maintain encrypted sessions, and one or more renderer engines associated with the user interface. There are multiple browser applications that are popular in contemporary use; viz.,
Internet Explorer (Microsoft), Chrome (Google), Safari (Apple), Firefox (Mozilla) and Opera (Opera). In this paper, we study browser applications that have a multi-threaded implementation.

In multi-threaded browser applications, there is exactly one parent browser process that manages the individual sessions using threads, irrespective of whether they were initiated using browser (application) windows or browser tabs. Therefore, with regard to tracing back the path of these browser sessions to the respective browser window or tab, it is sufficient if we can establish a one-to-one relationship between the said window or tab and the session that is supported. We leverage the fact that there is exactly one process through which the information passes from the network to the application.

The Mozilla Firefox browser, being a multi-threaded browser application, contains a single parent process and spawns off a thread for each sub-operation. 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, irrespective of the number of Firefox browser windows or tabs, there exists only one process\(^1\) 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, which do not adopt this approach. Furthermore, from an implementation standpoint, the Firefox browser parent process executes purely as a 32-bit process irrespective of the underlying hardware platform; and consequently, the addressable space for the entire browser application is limited to 4 GB, to be managed across multiple sessions.

Notwithstanding the existence of multiple browser threads processing simultaneous sessions, the web requests and responses become interleaved when they are logged by the application; the logged events appear as a single session since the application does not log information relating to the number of simultaneously active sessions. But a systematic analysis based on our methodology will breakdown this apparent singular session in to the respective tab sessions, as is described in the sequel.

IV. MAPPING BROWSER ACTIONS DURING ACTIVE BROWSER SESSIONS

A browser process, when activated (an active session is initiated by a user), localizes itself with respect to its network. This activity requires a series of message exchanges between the local host and the network it belongs to. Typically, this involves a link-local host localization that is achieved at layer-2 of the network stack. Following 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. Such sessions can be TCP or a secure TCP, i.e., serving HTTPS on server port 443. The number of sessions is dependent on the original server response and can vary depending on context. When multiple TCP sessions are in play, a browser can maintain multiple such sessions, using multiple network ports on the browser’s host machine – and each session is maintained until the browser host sends a FIN request to terminate that session. During a browser session, a user may make additional web requests until such time the user terminates the browser window or tab.

A. Browser Sessions & Logging

At any time, an active browser session requires the browser host machine to identify its position (ARP request-responses) within the network before requesting for information regarding the server (DNS request-responses) from where a service is requested. The network packets transmitted and received during such this period (ARP-DNS-TCP/UDP-HTTP) can be sensed and their network packet attributes can be used as parameters associated with the respective browser session. Consider a sequence of network transactions that take place when a user attempts to download a resource from the Internet:

Initially, the user makes a web request through some 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. Once the DNS response is received, the browser host machine initiates a TCP connection with the particular server and then makes a request for the resource. This request may or may not include a 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. Once 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 as well as network based communication activity leave enough trail information regarding all activities and the associated events. This information can be gainfully analyzed as required during reconstruction.

The most significant aspect of the browser session reconstruction is that neither the browser log file nor the network packets captured record any information pertaining to the specific browser window or browser tab that was responsible for generating a web session. For each browser event which 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 the server belongs to, e.g., Microsoft.com, Google.com, etc., the timestamp corresponding to the response in UNIX time format, and the structure information relating to how the response will be rendered on the browser. As a result a complete & rich web session on a browser window is represented in the form of a textual log entry. Normally, logs do not record the number of active browser windows or tabs and therefore reverse tracking has remained a challenge.

Xie et al [12] propose a method called Resurf to reconstruct web browser activity by pruning a referer based

\(^1\) Terminating the parent process terminates all browser sessions.
graph. ReSurf aims to reconstruct web surfing activities from traffic traces by analysing the referrer headers. Neasbitt et al [11] proposed a system named ClickMiner for reconstructing user-browser interactions from network traces. The ClickMiner uses referrer-click inference to prune a user browser activity graph based on referrer re-directions to ascertain those points where a user actively participated in generating new web requests. Through a user study, they demonstrated that Click-Miner can correctly reconstruct between up to 90% of user-browser interactions. Raghavan and Raghavan [7] present an approach to identify the source of a downloaded resource based on the relationships that are established using browser logs, network packets and file metadata. In that work, the authors demonstrated the use of metadata based associations to determine relationships between different sources of digital evidence, viz., user file system, browser logs and temporary internet files to determine the origin 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 / affected the file during a user’s online sessions. Using metadata associations, the authors determined file-to-file, file-to-log event, log event-to-log event relationships which is then traced to the source URL of the file download. We extend the work presented in [7] to identify causal relationships between the event sources; and in particular, we use the network parameters and the browser history logs to distinguish as well as identify concurrent events across sessions and coherent events belonging to a single session while handling multiple simultaneous browser sessions for a given browser application.

B. Tracking a Browser Session Using Browser History and Network Packets

When a browser interprets a web response and displays the corresponding HTML page, there are many aspects of this response that 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., the URL, the domain corresponding to the URL, the date and time when the web response was received, the name of the referral server2, number of visits in a given duration, the user under whose account the access was logged and so on. One must have noticed that browser windows and tabs are interchangeable in most browser applications. The specific window or tab that generated a specific web response is not recorded or logged. As a consequence, the entries on a browser history log file will appear in a sequential manner. Moreover, the web request that was responsible for the specific response is transparent to the utility that logs browser history information. The “missing” relationship information must be obtained through the network packets log.

When a packet capture utility captures network packets entering and leaving a host machine, it logs packets based on protocol wherein the utility uses specific filters to recognize and log network packets according to the protocol used in communication. If we consider a simple web request leaving such a host machine, it would require that machine to send out ARP to identify the gateway in a given network and then request for address resolution of the server the browser is attempting to communicate with. Once the address resolution yields an IP address, the browser attempts to set up a TCP channel with that server after which the said request is sent out. We have shown this sequence using a space-time (S-T) diagram in Fig. 1.

We note that there are a large number of packets entering and leaving 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, it terms of the data that can be traced back to a specific browser window or browser tab, the HTTP request contains insufficient information. Therefore, it is essential that any proposed method relies on a sequence of transactions that exhibit the characteristics depicted in Fig. 1 rather than a single HTTP request. We associate each browser response on the browser history log with a sequence of network packet transactions that correspond to the web request made by the browser on the host.

![Fig. 1. Space-Time diagram for making a web request](image)

We note that while the browser application tracks the web responses that are recorded sequentially, the availability of the network packets becomes necessary to decipher the web request. Besides, the packet capture must be so timed that it coincides with the browser sessions in question. This necessitates that the capture utility be placed so as to have complete visibility to the sequence of network packets3 exchanged in the same sequence as illustrated in the space-time diagram in Fig. 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 observed to be launched and filtered for network packets initiated by the host in question.

---

2 This entry refers to the server, if it exists, that redirected the browser to the specified URL.

3 In our experiments, to achieve this level of visibility, we used a packet capture utility on the browser host to track all incoming and outgoing network packets and filtered those that belonged to the browser application.
V. ELICITING SESSION BASED RELATIONSHIPS USING METADATA ASSOCIATIONS

The browsing application history is the information that a web browser application remembers and stores on the browser host machine as one browses the web. This includes information that one had entered into forms, passwords, and sites that one had visited. The browser history log records the requests sequentially; in the presence of multiple tabs, subsequent requests across tabs will be interleaved as the browser continues to generate additional web requests. In such a scenario, network packets are co-located in time, but belonging to (servicing) different browser sessions. Interesting thing to note here is 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 the browser log event with its corresponding sequence of network packets necessitates that one carefully identify the relevant characteristics of a browser log entry that can aid in this activity. The generic layout of a browser history log entry [4] is shown in Fig. 2 and we highlight the metadata we focus on from its list of attributes.

![Web browser log entry layout](Image)

Fig. 2. Generic layout a browser history log entry highlighting relevant metadata

On each network packet, there are many useful attributes (or metadata) that we may use to associate network packets with a particular network session: these can be the source and destination IP addresses, the protocol of communication, the timestamp corresponding to when a packet was seen leaving or entering the browser host machine, the host browser (TCP) process port and the server port numbers, and the session sequence number. The generic layout of a network packet [9] is shown in Fig. 3 and we highlight the metadata we focus on from its list of attributes.

Our approach identifies specific network packet sessions in accordance with the S-T sequence (shown in Fig. 1) and constructs a higher-level transaction that results from the web request generated by the browser process. This was argued to have a one-to-one correspondence with the browser history log entry as discussed in the previous section.

The distinction across simultaneous browser sessions (using multiple windows or tabbed sessions) is not apparent, when either the browser logs or the network packet trace, are seen in isolation. In contrast, the browser logs and network packets together provide a session context that our approach leverages to elicit the relationship between the said two information sources. We extract this relationship leveraging our knowledge of the sequence of actions that govern a browser based interaction (S-T diagram) and the timing information associated with the network packets that are exchanged between the host and the respective web servers that the browser application on the host is interacting with. This is discussed further in the remainder of this section.

![Generic layout a network packet as obtained from a packet capture utility highlighting relevant metadata](Image)

Fig. 3. Generic layout a network packet as obtained from a packet capture utility highlighting relevant metadata

A. Modeling Browser sessions on multi-threaded implementation

We identify a browser suiting our specifications (for example, the Mozilla Firefox browser) and fix the home page to a basic static page, e.g., Google.com. This ensures that when the home page loads, it consumes minimum additional memory and load time and it allows us to focus on the dynamics of the browser rather than on the web page. We prepare a list of web pages which contain of mixture of static pages as well as dynamic pages with animation and plugins involved. In sequence, we open additional tabs on an open browser window with each of these pages, opened one at a time and each time, we note the increase in memory consumption, the network ports used, number of additional network connections established and the memory locations used in memory by the multi-threaded browser process. This provides us with an estimate of how much additional memory is consumed when one of these pages are opened as a second tab, where the first tab is always set to the home page. This exercise is then repeated on a browser window for each of these web sites. In each case, the additional tab or window opened that directs to the homepage is used as a control.

As a next step, keeping two fixed tabs, we repeat the exercise outlined above for each of the websites in the list and note the memory consumed, the network ports used, the number of additional network connections established and the memory locations used by the multi-threaded browser process. This is then repeated for the browser windows in the same manner. Iteratively, we step through from two tabs to eight tabs and make notes of how the browser behaves as well as the parameters we outlined operate in each case. The case is also repeated for browser windows starting with two and increasing them in steps to eight and noting the parameters for each case. Based on the tasks carried out, we formulate a representation for a browser session.

On a single browser session, the network packets are causally related with respective browser events which in turn are structurally tied to that session – this can include the use of a specific process \( p \) and/or network ports from a set \( \{n_1, n_2, \ldots\} \).
to send and receive the network packets corresponding to the browser web request and its corresponding server response. Naturally, all communication sent and received for that session is maintained through the network ports binding it structurally to that browser session tab.

B. Developing a state space for a browser session

Consider an active browser session: it is the product of a web request $q$, a browser host forward network session $np_{	ext{fwd}}$, server response $r$, and a browser host return network session $np_{	ext{ret}}$. The network sessions $np_{	ext{fwd}}$ and $np_{	ext{ret}}$ are maintained by some browser process $p$ on network port $n$. Thus, a simple browser session with a single web request followed by a single server response can be represented by the set $s$ in eq. (1).

$$s = \{q, np_{	ext{fwd}}, r, np_{	ext{ret}}, p, n\} \quad \ldots (1)$$

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

The sequence of network packets transmitted from the browser host machine can be grouped into the set $NP_{fwd}$ while the sequence of network packets received at the browser host machine from the server is grouped as $NP_{ret}$. The timing information associated with the session information is quite critical in the reconstruction. This necessitates the association of time with the progress of a session that can be utilized during the reconstruction. In our work, while time is not explicitly represented in eqn. (2), we remind the readers that every request and response observed on the network is implicitly associated with a timestamp. As a consequence, both $NP_{fwd}$ and $NP_{ret}$ have elements (individual requests and responses seen on the network) that are mapped one-to-one on the timeline. Therefore, the browser session $S$ maintained by browser process $p$ on network port $n$ over a time period $T$ can be represented by the set $S$ in eqn. (2).

$$S = \{Q, NP_{fwd}, R, NP_{ret}, p, n\} \quad \ldots (2)$$

Using this representation for the state space of a single browser session, we define two relationships to elicit the relationships that define a coherent session and distinguish it from a concurrent session. A coherent session is one where the activities tagged correspond to a single browser session. A concurrent session is one where the activities tagged co-occur in time and they belong to distinct browser sessions without any further relationship. We define these relationships based on metadata value matches. A metadata value match can be defined to be an exact value match or a threshold-based match for some pre-determined threshold $\delta$ to accommodate for delays in the network that are beyond one’s control. We use these relationships to group related artifacts from the recorded evidence and reconstruct parallel browser sessions across multiple browser tabs.

C. Coherent event Relationship

When metadata matches occur across two artifacts $a_1$ and $a_2$ in the state space $\{Q, NP_{fwd}, R, NP_{ret}, p, n\}$ of a browser session, where $a_1$ and $a_2$ belong to the events from a browser log or from observed network traffic originating from the same server (domain), we define a coherent event relationship indicating that the two artifacts $a_1$ and $a_2$ belong to a single browser session. The relationship is denoted by $R_{coh}$ and expressed as $a_1 R_{coh} a_2$. By definition, this relationship is reflexive and associative. Therefore,

1. $a_1 R_{coh} a_2 \Leftrightarrow a_2 R_{coh} a_1$
2. $(a_1 R_{coh} a_2) \land (a_2 R_{coh} a_3) \Rightarrow (a_1 R_{coh} a_3)$

When multiple such artifacts $(a_1, a_2, a_3, \ldots a_k)$ exhibit an identical association between each other, we represent this relationship as $R_{coh} (a_1, a_2, a_3, \ldots a_k)$.

D. Concurrent event Relationship

When a metadata match occurs across two artifacts $a_1$ and $a_2$ in the subset of state space given by $\{R, NP_{ret}\}$ in a given time interval $T$, where $a_1$ and $a_2$ belong to the events from a browser log or from observed network traffic packets, we define a concurrent event relationship indicating that the two artifacts $a_1$ and $a_2$ share concurrency in time but belong to distinct browser sessions. The relationship is denoted by $R_{con}$ and expressed as $a_1 R_{con} a_2$. By definition, this relationship is reflexive but not associative. Therefore,

1. $a_1 R_{con} a_2 \Leftrightarrow a_2 R_{con} a_1$
2. $(a_1 R_{con} a_2) \land (a_2 R_{con} a_3) \Rightarrow (a_1 R_{con} a_3)$

When multiple such artifacts $(a_1, a_2, a_3, \ldots a_k)$ exhibit an identical association between each other, we represent this relationship as $R_{con} (a_1, a_2, a_3, \ldots a_k)$. When this condition holds, this can be interpreted, as evidence of at least $n$ distinct browser sessions, since any two artifacts, taken two at a time, will exhibit the concurrency relationship.

With regard to ordering and priority of the relationship thus defined, the coherency relationship supersedes the concurrency relationship, where applicable – consequently, if there exist two artifacts $a_1$ and $a_2$ such that they exhibit concurrency relationship as well as coherency relationship, then coherency is given priority and concurrency is dropped. This condition is necessary to account for those artifacts that belong to the same session and yet may be recorded in parallel TCP sessions that a browser may engage in. Our algorithm to reconstruct browser sessions is shown in Fig. 4.

---

4 This can be achieved in practice by installing a packet capture sensor that records all incoming and outgoing network packets from the particular browser host machine.
**Rachna algorithm**

*Given:* Browser request list \( Q = \{q_1, q_2, q_3, \ldots\} \),

Server responses \( R = \{r_1, r_2, r_3, \ldots\} \),

Observed network traffic at the browser host \( NP_{\text{sub}}, NP_{\text{rel}} \),

browser process \( p \), network ports \( n_1, n_2, n_3, \ldots \).

*Reqd:* identify number of multiple simultaneous browser sessions and relate each session with corresponding \( \{Q, R\} \)

BEGIN algorithm Rachna

Initialize \#sessions ← 0

Repeat

If server response \( R \) NOT NULL \#sessions ← \#sessions + 1

For each response in \( R \), do

(i) Parse the response \( r (r \in R) \) to determine list of referenced resources. Let this be list \( l \)

For each resource item in list \( l \)

(a) Determine the origin server \( s^* \) using metadata associations defined in [7]

(b) Relate the resource in \( l \) to TCP sessions in the network

(ii) Form the state \( \{q, np^\text{host}_{\text{sub}} r, np^\text{host}_{\text{rel}} n\} \) for server response \( r (r \in R) \) for each unique network port \( n \)

end for

\#sessions ← \#sessions + 1

Group all requests \( Q \) corresponding to responses in list \( l \) for a single session to derive the session state \( \{Q, NP_{\text{sub}}, R, NP_{\text{rel}} n\} \) for browser process \( p \) for each unique network port \( n \).

Stagger the session states so formed and order chronologically with respect to the web requests in \( Q \) for each session state.

Until (\#sessions = Cardinality of largest set \( | R_{\text{rel}}(\ldots) | \))

Display \#sessions as the number of simultaneous browser sessions.

For each distinct session, display corresponding set \( \{Q, NP_{\text{sub}}, R, NP_{\text{rel}}\} \).

END algorithm Rachna

**Fig. 4.** Browser session reconstruction algorithm

We call our algorithm **Rachna** which means – to form or construct – in Sanskrit. From the algorithm described in Fig. 4, the browser history log events show the web response as presented by a browser window/tab. However, the request that led to the response being generated remains implicit. Naturally, during reconstruction it becomes necessary to trace its origin. We derive these requests by parsing the server response to determine the list of resources, which are referenced and identify their respective source of origin.

For each resource, we correlate the network sessions initiated by the host within its network and group these request-response sequences together for form \( \{q, np^\text{host}_{\text{sub}} r, np^\text{host}_{\text{rel}} n\} \) for browser process \( p \). Having identified such the sequence, we update the \#sessions which tracks the number of simultaneous sessions that are deployed. The method to reconstruct multiple browser sessions by applying the **Rachna** algorithm is illustrated in Fig. 5.

The network packets that are different colored indicate the concurrency relationship between them that is identified based on metadata associations. When the network packets demonstrate a coherency relationship with a particular server response belonging to the set \( R \), they are grouped together to indicate the session coherency. The entities grouped thus are then used to reconstruct each coherent session.

**VI. IDENTIFYING RELATIONSHIPS IN BROWSER ARTIFACTS USING METADATA ASSOCIATIONS**

We executed the following experiment. A user was asked to use the Mozilla Firefox browser (with developer mode and network request tracker enabled) and conduct a specific 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. On the first window, the user opened connections to securecyberspace.org and the online TV news guide sidereel.com. On the second browser window, the user logged into his personal yahoo email account and the second tab was connected to the online video streaming server youtube.com. Subsequently, the user separated the browser tab connected to yahoo email as a separate window.

On the browser tabs connected to sidereel.com and

5 Open browser and hit function F12 on the keyboard
We adapted the AssocGEN analysis engine [8] to identify coherency and concurrency relationships based on the metadata determined from the browser log entries and network packets obtained during active network sessions. The browser history log entries and the individual network packets, associated via their respective parameters identified in Fig. 2 and Fig. 3, are grouped together depending on the relationships exhibited, i.e., those simultaneous browser tabbed sessions that exhibit coherent event relationships on artifacts belonging to a single browser tabbed session exhibit both causality as well as session dependence (browser process to port binding) across browser log entries and network packets. This is determined and the different simultaneous sessions are separated.

Initially, #sessions is set to 0 and since the server response is not NULL, we update #session to 1. We parsed the original HTML response for each session to obtain a total of 1489 resource requests spread across five browser sessions and over the 52692 network packets captured for the duration of the entire browsing, as summarily listed in Table VI. To eliminate requests generated from the browser during the rendering process, we used the metrics provided by the developer mode in the Firefox browser to isolate the HTML content from the remaining elements such as JavaScript, JPG, CSS, XML and so on. For each reference resource that was downloaded subsequently, we identified the origin server. Each time a response from a new server was identified, the variable #session was incremented. Once the origin server for each resource was identified, we discovered coherency relationships between the corresponding server response and the packets from network traffic. The set of all coherent artifacts (server response \( r \), resource list \( l \), and network traffic \( n_{epoch}^{host} \), \( n_{epoch}^{net} \)) were grouped together under a single browser session. As mentioned in the algorithm, each referer 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 allowed us to distinguish between the main elements of the page and the add-on elements. Additionally this distinction helped us to link-and-chain server responses from a streaming server (i.e., youtube.com) where a single request can contain a response that in turn initiates a new request.

While establishing coherency between the artifacts from a single browser session, we compare the resource lists across the sessions with the aggregated network traffic at the browser host machine. This reveals concurrency relationships between the resources and the packets from network traffic. Then we determine the largest set of artifacts for which the concurrency relationship can be established, i.e., we determine the cardinality of the \( R_{coh}(...) \). The variable #sessions is updated for each new element that is discovered in this set. In our case, this value is five.

### VII. Results & Discussion

Once the artifacts are grouped according to the \( R_{coh} \) and \( R_{con} \) relationships, we aggregated the network traffic that serviced each web request sent out from the browser and determined there were 13 distinct web requests generated owing to user’s actions by the browser during the period of observation. Using the network traffic, we isolated the host to discover that the browser contained a total of five tabbed browser sessions. We identified two static page connections from the first browser window, i.e., google.com and securecyberspace.org. These connections did not generate repeated reconnections and we determined that these sessions were associated with a single browser window. This is determined based on the memory locations allocated to that browser window (the first window) which contained the session context in memory for browser process p. However, the user subsequently requested for a page called “Practicing Security” on the page connected to securecyberspace.org. No further activity was observed on these pages. The tabbed sessions connecting to the online media servers, i.e., siderel.com and youtube.com, respectively were discovered to generate the most number of rendered requests, 189 resource requests for siderel.com and 158 resource requests for youtube.com. The pop-up and dynamic prompts are not included in this count. During our analysis, these request elements did not provide a precise source of origin and therefore the ambiguity. The connection with the user’s email server also had a large number of such requests, although this was restricted to the server’s home page. Once the user’s request progressed to the mail login pages, this drastically reduced in number. We noted that this is primarily the advertisements provided on the website that contain referred elements from the parent server (i.e., au.yahoo.com) which generates new connections on behalf of the server. We note that based on further analysis of the browser cache and stored information on the host machine, we were also able to identify the user’s stored user credentials for the email login. This can be a useful function during the process of browser session reconstruction to recover user details and identify which particular account a user accessed. Both the email server as well as the media server youtube.com repeatedly refreshed their page contents identified in the page elements such JavaScript, JPG, XML, and some ActiveX content that generated new connections on the host-server connection on new network ports.

<table>
<thead>
<tr>
<th>Tabbed Session ID</th>
<th>Timestamps</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13-08-2015 PM 02:52:15</td>
<td><a href="http://www.google.com/">www.google.com/</a></td>
</tr>
<tr>
<td>2</td>
<td>13-08-2015 PM 02:52:37</td>
<td>securecyberspace.org/</td>
</tr>
</tbody>
</table>
The reconstructed browser sessions (server responses R) are tabulated in Table II. We validate our by repeating this activity on the same browser providing the URL as identified by the browser history log for each tabbed session and comparing the records generated. We use analysis engine to first process the browser history logs and load the log records and their metadata (after parsing) into the engine’s repository. Once the browser logs are completely traversed, we traverse the network packets obtained during the capture. Thereafter, we use a procedure to generate all object relationships based on associations identified in their metadata across the browser history log entries, TCP connections including the process information and network port information from memory and the network packets in the packet capture.

Once the associations and generated and stored into the repository, we elicit syntactic relationships between artifacts of the same type (i.e. among network packets and browser history logs respectively) and semantic relationships across the types to discover the origins of the web sessions concerned. Artifacts that belong to the same application are determined to have coherent event relationship Rcoh. This is typically true of all records from a browser history log or between network packets.

The concurrent event relationships Rcon are determined to exist between artifacts that occur at the same time but contain different session context, and belong to distinct browser sessions. This is true of browser history records that are captured across tabs running simultaneous sessions or between network packets that service these parallel sessions across different browser tabs. Based on the reconstruction, we replayed the sessions on a Firefox browser to corroborate the evidence tested and the snapshot of the replayed session based on reconstruction is shown in Fig 6. We note that while the algorithm was able to distinguish the sessions were carried out across two browser windows and over five tabbed sessions, the algorithm was did not discern the change from two browser windows expanding into three windows during the session. We believe 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. We are currently exploring this aspect of the browser behavior.

**A. Results**

The reconstructed browser sessions (server responses R) are tabulated in Table II. We validate our by repeating this activity on the same browser providing the URL as identified by the browser history log for each tabbed session and comparing the records generated. We use analysis engine to first process the browser history logs and load the log records and their metadata (after parsing) into the engine’s repository. Once the browser logs are completely traversed, we traverse the network packets obtained during the capture. Thereafter, we use a procedure to generate all object relationships based on associations identified in their metadata across the browser history log entries, TCP connections including the process information and network port information from memory and the network packets in the packet capture.

Once the associations and generated and stored into the repository, we elicit syntactic relationships between artifacts of the same type (i.e. among network packets and browser history logs respectively) and semantic relationships across the types to discover the origins of the web sessions concerned. Artifacts that belong to the same application are determined to have coherent event relationship Rcoh. This is typically true of all records from a browser history log or between network packets.

The concurrent event relationships Rcon are determined to exist between artifacts that occur at the same time but contain different session context, and belong to distinct browser sessions. This is true of browser history records that are captured across tabs running simultaneous sessions or between network packets that service these parallel sessions across different browser tabs. Based on the reconstruction, we replayed the sessions on a Firefox browser to corroborate the evidence tested and the snapshot of the replayed session based on reconstruction is shown in Fig 6. We note that while the algorithm was able to distinguish the sessions were carried out across two browser windows and over five tabbed sessions, the algorithm was did not discern the change from two browser windows expanding into three windows during the session. We believe 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. We are currently exploring this aspect of the browser behavior.

**B. Discussion**

In this paper, we have developed a method to distinguish multiple simultaneous sessions based on artifact relationships derived from metadata-based associations. We use coherency relationship to establish connectedness between artifacts belonging to single session and use concurrency to distinguish between artifacts that share time of occurrence but not session context. However, there are some special cases that we need to consider; we have been operating under the assumption that there are multiple browser sessions at the initiation point.

We may also need to consider cases where a new tab is created along the way and detect them to effectively use the algorithm developed in this paper; particularly so when the new session communicates with the same server as the immediate previous tab. If we consider this case, while we can map the resources from the server after parsing the original server response as described in the algorithm, we should take cognizance of the fact that 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 we map the resources list against the network traffic. This can be identified by tracking the network ports on the host to determine when it requests for a new connection from a server which has an existing connection from browser process p. There is scope for some further refinements in the detection method for 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 contained and processed by the browser application is volatile and is contained in the browser host memory. Between the parent process and the threads maintaining multiple browser sessions, there is two-way communication that ensures 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 point of the request. All completed network services are generically returned through the network stack to host memory (shared memory implementation) wherefrom the process responsible for making the call reads the information for further processing. We also noticed that the algorithm could not deal with the pop-up windows and dynamic prompts with page timeouts. Since the precise origin of the pop-ups on a page could not be associated with a request originating from the host machine, the algorithm was unable to deal with this case.

While this 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 reconstruction in the context of investigating security incidents. Providing the attribution by ensuring two-way information recording can go a long way in developing attack resilient browsers on the morrow.

VIII. CONCLUSIONS & FUTURE WORK

In this paper, we demonstrated the feasibility of isolating multiple simultaneous browser sessions on multi-threaded browser applications based on the browser application logs and the network traffic logs on the browser host machine. We used “metadata based associations” to identify and reconstruct all tabbed sessions that are part of a typical browser interaction. We defined and used two relationships derived from metadata based associations, viz., coherency & concurrency among the artifacts derived from browser history logs and the network traffic to identify the number of simultaneous sessions. We developed the Rachna algorithm that identified these relationships to discover the number of simultaneous browser sessions deployed and reconstruct them. We demonstrated our method by conducting the analysis on a five-tabbed Mozilla Firefox browser using the browser history logs and the network packets exchanged. We derived the relationship 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 which request reconnections from the same server.

Of late many browsers have stated providing the “incognito” or “silent” option to avoid logging browser activities on the browser application logs. This can have a significant impact on the reconstruction method explored in this work and can impede the identification of the relationships defined. 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. In future, we intend to extend our approach to investigate such scenarios. Besides, we intend to explore the “minimum information” needed to reconstruct with acceptable accuracy. We observe that increasingly many active pages, particularly those associated with online media contain such pop-ups and dynamic prompts and we are currently working on identifying the distinguishing features of a pop-up on a web response and incorporating this feature into our algorithm in the future. We believe that such understanding may lead to fundamental changes in the specification of Internet related active devices and applications, in future.

REFERENCES