

Implications on Privacy and Forensics on Current Browsers

ABSTRACT: To date, the evolution of Web-related technologies has been mainly driven by the user's quest for ever faster and more intuitive **World Wide Web (WWW)**. One of the most recent stages of this evolution is built around the idea that the user experience can be further improved by enabling the browser to predict and preload Web resources the user is most likely to seek. While this concept of 'pre-browsing' is beneficial from the point of view of speed and performance, little thought has been put into understanding and analyzing its potential implications on 'other' metrics – including user reputation and privacy. In this paper, we first provide a comprehensive overview of the main pre-browsing mechanisms as implemented in/by today's browsers. Subsequently, we outline a few hypothetical (yet very real) scenarios in which these mechanisms end up turning the browser into a dangerous tool that acts against its very own user. The ultimate goal of our work is to make the wider Internet community critically rethink the way pre-browsing mechanisms are implemented and used in today's WWW.

CCS Concepts: Information systems → World Wide Web → Social recommendation. • Security and privacy → Systems security → Browser security. • Applied computing → Computer forensics → Evidence collection, storage and analysis.

Keywords: Pre-browsing; unsolicited Web requests; user privacy; user reputation; user forensics; HTML5; Chrome

1 Introduction

Only three decades into its existence, the World Wide Web (WWW¹) has become an essential and ubiquitous commodity of the modern age – alongside piped water and electricity. Nowadays, services and business that do not have an online presence are considered to be 'out of touch' with the mainstream society and are almost certain to struggle and/or fail [1]. Similarly, individuals without access to the WWW are at a serious disadvantage in almost every aspect of life that spans from learning, looking for jobs, shopping, to socializing.

With the ever growing importance and prevalence of WWW, we are becoming increasingly reliant on the use and performance of Web browsers – software applications that allow users to access, traverse and retrieve the WWW resources. And, while in the past Web browsers were almost exclusively built for and used on desktop and laptop computers, any device capable of connecting to the Internet (including mobile phones, smart watches, wearable tracking devices) are likely to host one or multiple Web browsers. In fact, the modern-day dilemma is not so much whether a Web browser should be available on an Internet-enabled device (regardless of its size and capability), but what can be done to make the performance of that browser faster and more user friendly.

Users' quest for ever faster and more intuitive WWW has been the driving force behind the evolution of Web-browser technology as well as numerous Web-related protocols. One of the most recent stages in this evolution is driven by the idea that a user's WWW experience can further be improved by predicting and/or preloading Web resources that are most likely to be sought by that particular user. To make this objective attainable, the following three mechanism/technologies have been introduced over the past several years:

a) *Instant search* (also known as *auto-complete*, *auto-suggest* or *type-ahead search*) is a feature presently supported by many Internet Search Engines (ISE), the most notable one being Google.com. With this feature, the moment the user starts typing in the ISE's search box, the most likely search terms are suggested to the user while the respective results appear in the ISE's main window. And, as the user continues typing in (i.e., keeps refining the search terms), the results get dynamically updated. The predictions/suggestions displayed by this feature are generally influenced by a number of different factors, including: the user's previous searches, the user's current geographic location, recent searches done by other people, etc. [2], [3]. (An illustration of instant search for 'hotels' conducted in Google ISE by a user located in Toronto is provided in Figure 1.) **According to Danny Sullivan**, the instant search feature

¹ Acronyms WWW and Web will be used interchangeably in this paper.

not only helps guide user searches and allows the searchers to see results without clicking, but it also saves 2-5 seconds per each conducted search, ultimately resulting in an overall faster WWW experience for the user [3].

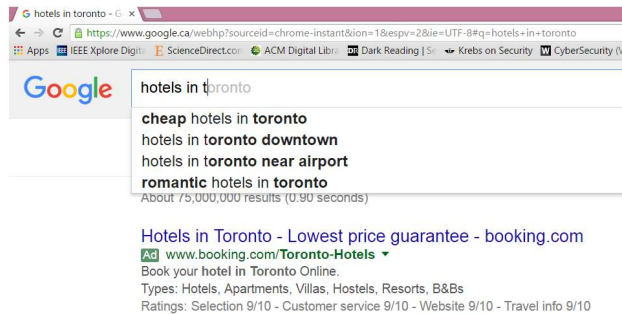


Figure 1. Instant search by a user in Toronto

b) *Resource hints* is a term used to refer to several different HTML5 features/options introduced to facilitate (i.e., speed up) the process of Web resource loading in a Web browser. In particular, the term covers four main categories of resource (pre)loading: pre-connect, dns-pre-fetch, pre-fetch and pre-render – all four being implemented as a relation (rel) type/attribute of HTML5’s Link Element <link> [4]. When found in a Web page (i.e., HTML5 document) resource hints are intended to instruct the browser to get hold of resources that are related to or are part of the most likely next-page navigation, ahead of time. Thus, if/when the user actually decides to request the given page, the respective resources will be simply pulled out from the ‘background’, giving an illusion of instantaneous (near zero-delay) retrieval. (We explain all four resource hints types in more detail in Section 3.)

c) *Predictive browsing* (also known as *predictive URL*, *predictive services* or *instant search*) is a feature initially introduced by the Google Chrome browser (circa 2013), and subsequently implemented by several other browsers. The goal of this feature is to assist the user by speeding up the process of typing in a desired URL and retrieving this URL’s respective resources. Namely, as soon as the user types in the first few characters of a desired URL in the Omnibox (see Figure 2); the browser starts providing suggestions of URLs that the user is most likely to request. These suggestions are generally based on the statistics derived from the user’s past navigation history as well as on the statistics about most common URLs requested by other users. In cases when the hit rate (i.e., the confidence that a predicted URL will be actually requested) is high, the browser may also initiate a DNS lookup, TCP pre-connect and even full pre-render of the URL’s respective Web page in a hidden tab [5]. In its essence, *predictive browsing* combines the ‘autocomplete’ functionality of *predictive search* with the preloading power of *resource hints*. (We will describe the details of Chrome’s predictive browsing in Section 3.)

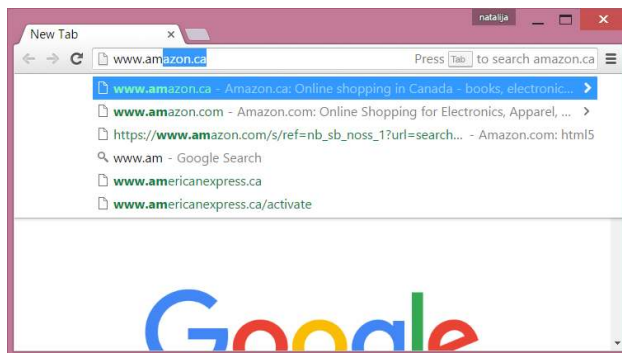


Figure 2. Predictive browsing in Chrome

Now, an average Web user is likely to consider all three of the above described mechanisms (*instant search*, *resource hints*, *predictive browsing*) useful, as they undoubtedly have the potential to facilitate faster browsing experience. As a result, in many browser types, including Google Chrome², *resource hints* and *predictive browsing*

² According to StatCounter Global Stats, for over 70% of WWW users, Google Chrome is the browser of choice. Hence, most of our discussion will revolve around this particular browser. As for the other browsers types, Firefox is used by 16%, Internet Explorer by 5%, and Safari by 3% of Internet users [7].

are enabled ‘by default’³. This - combined with the fact that users generally tend to keep the default settings of their applications unchanged [8] - further implies that *resource hints* and *predictive browsing* are likely to be found in a significant number, if not the majority, of browser instances currently used in the Internet. (In the reminder of this paper we will refer to *resource hints* and *predictive browsing* as *pre-browsing* mechanisms.)

While we do not intend to question the practical usefulness of *pre-browsing* mechanisms from the performance/speed point of view, the work presented in this paper seeks to address the potential negative implications of their use on the privacy and reputation of an ordinary Web user. Namely, the *pre-browsing* mechanisms are generally designed to be executed without the user’s direct involvement (i.e., knowledge or approval) and in an obscure ‘behind the scenes’ manner. And even though this un-intrusiveness has its obvious advantages when it comes to speed and convenience, it can also be easily misused (both intentionally and unintentionally) by turning a browser into a dangerous tool that acts against its user. Hence, the goal of our work is to bring awareness to these possibilities, and to cause the wider Internet community to rethink the way *pre-browsing* mechanisms are implemented and used in today’s WWW.

The content of this paper is organized as follows. In Section 2, we discuss the significance and implications of using IP addresses as a means of identifying and tracking WWW users – a common Internet practice that is a precursor to the problem discussed in this paper. In Section 3, we provide a detailed overview of the two *pre-browsing* mechanisms – *resource hints* and *predictive browsing*. In Section 4, we present the results of our experimentation concerning the performance and implications of *pre-browsing* mechanisms in Google Chrome. In Section 5, building on the results from Section 4, we outline several real-world scenarios in which *pre-browsing* mechanism have the potential to negatively impact the user’s reputation and privacy. In Section 6, we close the paper with conclusions and recommendations for future research.

1. BACKGROUND

1.1 Relationship between a User and His Computer/Browser Namely, as soon as the user types in the first few characters of a desired URL in the Omnibox (see Figure 2); the browser starts providing suggestions of URLs that the user is most likely to request. These suggestions are generally based on the statistics derived from the user’s past navigation history as well as on the statistics about most common URLs requested by other users. In cases when the hit rate (i.e., the confidence that a predicted URL will be actually requested) is high, the browser may also initiate a DNS lookup, TCP pre-connect and even full pre-render of the URL’s respective Web page in a hidden tab [5]. In its essence, *predictive browsing* combines the ‘autocomplete’ functionality of *predictive search* with the preloading power of *resource hints*. (We will describe the details of Chrome’s predictive browsing in Section 3.)

As today’s world grows ever more reliant on the WWW, the boundaries between humans and their respective Internet-enabled devices and browsers are becoming increasingly blurred. Namely, in many disciplines it has become a common practice to assume that a user’s device and browser are nothing but a mere extension of the user, and their only mission is to carry out the tasks explicitly requested by the user. Consequently, (in all but cases of verifiable device/browser hacks) the user may be considered fully accountable for actions or requests executed by his device/browser.

Browser histories are one mean of tracking the user behavior to make a prediction and browsing logs are used to mine what Google terms as “to improve browsing experience”. A web browser such as Firefox or Internet Explorer will load the current user’s browser history into memory and compare each link (anchor) on the page against the user’s previous history. Web browsers store a list of web pages in local history for about a week by default. Your browsing history improves your browsing experience by autocompleting a URL in your address bar, helping you search for previously viewed content, or coloring previously visited links on a page. The internet is a vast and extremely elusive place. Trying to get a handle on what someone does when they are browsing the internet is a difficult task, especially when individuals now tend to clear browsing data from their browsers.

Web search logs, the use of data stored in transaction logs of Web search engines, Intranets, and Web sites can provide valuable insight into understanding the information-searching process of online searchers. This understanding can enlighten information system design, interface development, and devising the information architecture for content collections. DuckDuckGo (DDG) is an Internet search engine that emphasizes protecting

³ Note, instant search is a feature of an ISE not a browser, hence not mention here.

searchers' privacy and avoiding the filter bubble of personalized search results. DuckDuckGo distinguishes itself from other search engines by not profiling its users and by deliberately showing all users the same search results for a given search term. DuckDuckGo emphasizes getting information from the best sources rather than the most sources, generating its search results from key crowdsourced sites such as Wikipedia and from partnerships with other search engines like Yandex, Yahoo!, Bing, and Yummly. Tor directs Internet traffic through a free, worldwide, volunteer network consisting of more than seven thousand relays to conceal a user's location and usage from anyone conducting network surveillance or traffic analysis. Using Tor makes it more difficult for Internet activity to be traced back to the user: this includes "visits to Web sites, online posts, instant messages, and other communication forms". Tor's use is intended to protect the personal privacy of users, as well as their freedom and ability to conduct confidential communication by keeping their Internet activities from being monitored.

The concepts of *user tracking* and *Web-related forensics* are perhaps the best illustration of how tight the 'coupling' between users as persons and their device/browser is. For example:

- In *user tracking*, the IP address and cookies⁴ associated with a user's device (i.e., browser) are used to identify that particular user in the 'on-line world'. Subsequently, all observed Web requests that happen to carry those particular IP address and/or cookies are assumed to be generated with full knowledge and intent of the given user and, as such, are used to track the user's online behavior as well as gauge their interest in different product and services [9]. User tracking mechanisms put relatively little (if any) effort in distinguishing between genuine user requests and those that were automatically generated by the user's browser.
- The goal of *Web-related forensics* is to gather information about which Web sites and files a user has accessed while browsing the WWW, in order to prove or disprove a claim of misconduct. The places where this information is typically collected from are: a) the browser history and cache on the user's device (if accessible), and/or b) the log files of the edge gateway that connects the user to the Internet, and/or c) the log files of Web server(s) hosting the disputed files. If any evidence of the disputed files being accessed through the user's device/browser (while in the user's possession) is found in either a) or b) or c), the user himself could be held responsible – even without an explicit proof that the user, not the browser, was the one who actually initiated those requests.

General research related to Web browser forensics has been targeted to specific Web browsers or to structural analysis of particular log files. Jones (2013) explained the structure of the index.dat file and how to extract deleted activity records from Internet Explorer [14]. He also introduced the Pasco tool to analyze the index.dat file. After simulating an actual crime, he described the IE and Firefox 2 Web browser forensics in two different publications [15,16]. In Section 1, he introduced the Pasco and Web Historian tools for IE forensics, which are available to the public, and the IE History and FTK tools, which are not. In Section 2, he described forensics in Firefox 2 using a cache file. The cache file in Firefox 2 is not saved in the same way as in IE, so he suggested an analysis method using the cache file structure. Pereira (2009) explained in detail the changes in the history system that occurred when Firefox 2 was updated to Firefox 3 and proposed a new method of searching deleted history information using unallocated fields [17]. During execution of Firefox 3, a rollback journal file is generated using a small section or the entire contents of Places.sqlite. If processing is stopped, this rollback journal file is erased [17]. For this reason, it is possible to extract history information of Firefox 3 in unallocated field. The author suggests a method of extracting history information from Firefox 3 by examining the SQLite database structure.

The purpose of our work is to demonstrate that the pre-browsing mechanisms outlined in Section 1, in combination with our tendency to assume that devices and browsers are nothing but innocuous and trustworthy 'extensions' of their owners/users, can lead to a number of potential misuses. To lay a foundation for further discussion of this issue, in the subsequent sub-section we provide an outline of a typical WWW client-server architecture and its most significant elements and interactions as pertaining to our work.

⁴ IP address is a unique identifier assigned to every computer connected to the Internet. Cookies are small data files that a Web server stores on a user computer to keep track of that user's browsing. Between the two, the use of cookies is a preferred and more accurate mechanism of user tracking in the WWW. However, in cases when cookies are disabled on the client/user side, or are not deployed by the Web-site (according to W3Tech Web Technology Surveys, 50% of Web sites currently do NOT use cookies), IP addresses are used as an alternative means of user tracking [10].

1.2 Typical WWW Client-Server Architecture

The below figure outlines the most significant elements of a typical WWW client-server architecture and those include:

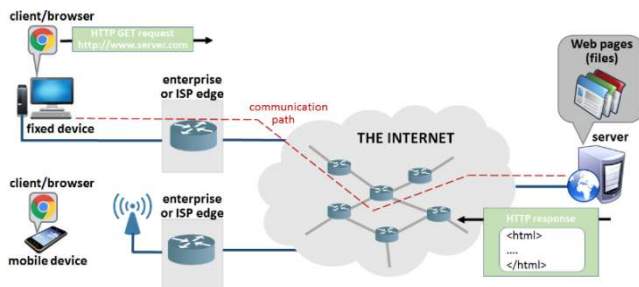


Figure 3. Typical WWW Client-Server Architecture

- The client, in the case of the WWW is a Web browser running on the **user's device**. The device could be either 'fixed' (e.g., a desktop computer) or 'mobile' (e.g., a laptop, tablet or smartphone), and is uniquely identified either with a static IP address (common scenario in fixed enterprise networks) or a dynamic IP address (common scenario in cellular and public WiFi networks).
- The edge network, which provides physical connectivity between the user device and the rest of the Internet. This could be either an enterprise edge network (e.g., when the device is used at work), or an ISP edge network (e.g., when the device is used at home). In either case, the edge network typically contains one or multitude of specialized devices which engage in monitoring and/or logging of the passing traffic (e.g., gateway routers, firewalls, **proxies, etc.**).
- The Internet core, which is responsible for routing packets, including those that carry client-server HTTP requests and responses, from their source to the intended destination.
- The Server, which in the case of the WWW is a machine capable of hosting and sharing Web-pages (i.e., files) over the Internet. Unlike Web clients, Web servers are generally assigned static IP addresses.⁵ **According to Netcraft 2016**, the Internet/WWW currently comprises millions of such machines [11].

Now, whenever a particular client requests a Web page from a particular server (by means of a GET HTTP request), various types of 'artifacts' related to this event get recorded at various points along the communication path between the two entities. For example:

- On the client side, the outgoing request (in particular the URL of the requested page) gets recorded in the *browser history*, while the resources that the requested page is made of get stored in the *browser cache*⁶ as **they arrive** from the server. As earlier indicated, browser history and cache are of great significance from the perspective of Web forensics, as they can help prove that a particular Web request has taken place. Nevertheless, the main challenge of relying on browser history and cache as forensics evidence is that they are owned by and directly accessible to the user, and as such could be easily modified or deleted (intentionally or unintentionally) or simply rendered unavailable if the user decides to deny access (in which case a search warrant is required to be able to access these resources).
- The given HTTP request is likely recorded, together with the traffic of other users, in the logs of the specialized devices in the edge network (gateway, firewall or proxy). It should be noted, however, that edge networks are not always mandated to record these logs, hence from the forensics point they may have limited practical relevance.
- The intermediate routers in the Internet core could also keep a record of the given HTTP request in their own traffic logs. However, due to the high volumes of passing/recorded traffic, these logs are generally kept for a very short interval of time. Consequently, their practical use as forensics evidence is rather limited, similar to ii).
- The server logs are the final place where the given HTTP request gets recorded⁷. In general, server logs have particularly important significance from the forensics point of view, for two main reasons. Firstly, most organization tends to retain their Web server logs over long periods of time. Secondly, in most organizations Web server logs are well protected and could only be altered by the site administrator. Hence, when a record of a Web request arriving

⁵ In reality the server is also connected to the core Internet through an edge network. However, for simplicity purposes, we omit this from Figure 3.

⁶ Browser history allows for faster identification and retrieval of previously visited Web pages, while browser cache allows for previously viewed pages to get re-visited without generating any new traffic.

⁷ Web server logs collect a wealth of data, including which specific pages/resources were requested, at what time, and from which IP address. This data is then used to deduce information about the overall number of visitors to the given site as well as to analyze their browsing behavior.

from a particular client/host (i.e., IP address) is found in these logs, it is impossible to deny the authenticity of the given event - unless one can prove that the logs were altered (e.g.) by a malicious site administrator.

With the above facts in mind, our work focuses on the following fundamental question: for an HTTP request generated by the client/browser and sent along the communication path outlined in Figure 3, is there a way of knowing whether the given request was generated as a result of an intentional action by the user, or perhaps it was generated without the user's knowledge and approval? In particular, we are set to examine whether the artifacts collected along the given communication path provide enough information to tell these two different types of requests apart.

2. PREBROWSING MECHANISMS IN HTML5 AND CHROME

In this section we provide a more detailed look at the *resource hints* and *predictive browsing* as examples of mechanisms that can trigger a browser to generate unsolicited HTTP requests.

2.1 Resource Hints in HTML5

HTML - short for Hypertext Markup Language - is a well-known and widely used interpreted tagged markup language that enables creation of Web pages (i.e., hypertext documents)⁸. An example of a simple Web page written in HTML is given in Figure 4. Originally introduced in 1990-ies, HTML has been evolving over the years so as to accommodate ever more diverse and sophisticated Web contents, as well as to deliver increasingly better and faster performance for the end-user.

```
<!doctype html>
<html lang="eng">
<head>
  <meta charset="utf-8">
  <title>HTML Tutorials</title>
</head>
<body>
  <h1>My First Web Page</h1>
  <h3>Welcome To My Website</h3>
  <p>Content Goes Here</p>
</body>
</html>
```

Figure 4. Simple HTML Web page

In the most recent version of the protocol (HTML5) a special new set of features/options have been introduced in order to support the idea of 'instant' (zero-delay) Web-page load. Namely, as pointed in [5], a browser that starts downloading a Web page only after the page has been explicitly requested by the user will inevitably result in substandard browsing experience that is riddled with various types of network delays⁹. The only way to spare the user from experiencing the browsing/network delays is by trying to anticipate their requests ahead of time, and then preload the most critical resources associated with those requests even before the actual 'click on the link' action occurs. That way, the resources will be readily available when the user actually requests them, giving an illusion of an instantaneous (zero-delay) download.

Now, the idea of 'instant' browsing is not entirely new. This concept was originally supported through the implementation of *Web-cache* – a memory location where the resources of previously visited Web-pages are stored, allowing that these resources be instantaneously retrieved whenever the user decides to revisit them. Unfortunately, as such, Web-cache is of no use when it comes to retrieving pages that have not been previously visited. Hence, to enable zero-delay browsing of pages that are to be visited for the first time, or pages that have been purged or expired from the cache, HTML5 has come up with a set of features commonly referred to as *resource hints*.

According to W3C Working Draft, there are four different types of resource hints provisions in HTML5 [4].

a) ***dns-pre-fetch*** is a resource hint that can be used to suggest a browser to perform a DNS pre-fetch (i.e., IP lookup) for a particular hostname. The following is a situation where this feature might be useful in practice. Imagine the user is currently visiting page_A.html hosted on server_1.com, and there is a high likelihood that the Web-page the user is going to visit next is page_B.html located on another server (server_2.com) – as illustrated in Figure 5. To

⁸ An interpreted markup language means that whenever a browser receives an HTML file, the browser interprets the file's markup elements and displays the results, hiding the actual markups from the user.

⁹ These delays include: DNS lookup delay, TCP handshake delay, SSL negotiation delay, delay to obtain base HTML page ... For more see [5] and [6].

expedite the loading of page_B.html (if and when the user requests it), we could place the below tag in the <head> section of page_A.html:

```
<link rel="dns-pre-fetch" href="//server_2.com">
```

That way, the browser would start performing the DNS lookup for sever_2.com right away (i.e., while the user is still viewing page_A.html), making sure that the IP address of server_2 is obtained even before the user actually clicks on http://server_2.com/page_B.html.

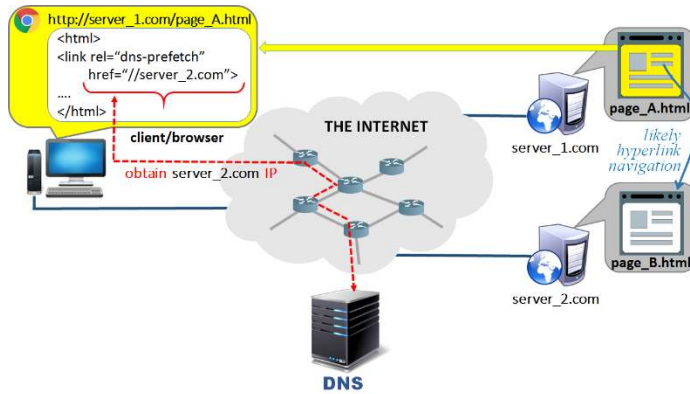


Figure 5. Linked pages hosted by different servers

b) **pre-connect** is a resource hint option that can be used to initiate an early connection with a Web-server, which includes the DNS lookup, TCP handshake, as well as optional TLS negotiation. As such, pre-connect clearly goes step further in minimizing/masking networking delays relative to *dns-pre-fetch*.

In the example of Figure 5, the following tag placed in the <head> section of page_A.html would prompt the user's browser to establish an early (pre)connection with server_2.com.

```
<link rel="pre-connect" href="//server_2.com">
```

Also, in the given example, the decision whether to use *pre-connect* or just *dns-pre-fetch* for server_2.com is (i.e., should be) closely tied to the actual probability that the user navigates to page_B.html from page_A.html. Clearly, the higher this probability, the more reasonable it would be to use the *pre-connect* resource hints option.

c) **pre-fetch** is a resource hint option that further builds on the functionality of a) and b). Namely, in addition to performing the DNS resolution and establishing a connection with a particular server, *pre-fetch* also allows that some resources (e.g., the base HTML file of a Web-page, images, JavaScript-s, CSS-s, etc.) be downloaded from this server ahead of time and stored in the browser cache. For example, in the scenario of Figure 5, the following tag placed in the <head> section of page_A.html would prompt the user's browser to download and cache the base HTML file of page_B.html – the key Web resource (and the first one to be retrieved) during the rendering of this page.

```
<link rel="pre-fetch" href="//server_2.com/page_B.html">
```

Clearly, by allowing whole parts of a page be obtained by the browser even before the page gets actually requested, *pre-fetch* enables even further reduction in networking/browsing delays. However, given the communication and storage overhead associated with *pre-fetch*, the use of this resource hints option is justified only in cases when the probability that the user actually navigates to a specific page is greater than in the case of a) or b).

d) **pre-render** is the most encompassing resource hints option - it allows not only that the base HTML file and all other components of a page get preloaded ahead of time, but also that the page itself gets fully laid out, its respective CSS-s applied and JavaScript-s executed. Put another way, it is as if the page is open in a hidden tab, and the moment the user navigates to the page's URL, the hidden tab is immediately swapped into view [5]. As such, *pre-render* is the only resource hints option that can truly cut the browsing delay down to zero, giving an illusion of truly instantaneous browsing.

In the scenario of Figure 5, the following tag placed in the <head> section of page_A.html would prompt the user's browser to pre-render (i.e., preload and preassemble) the entire page_B.html.

```
<link rel="pre-render" href="//server_2.com/page_B.html">
```

Now, it should be pretty clear that out of all four resource hints options, the use of *pre-render* is associated with the most significant communication, storage and processing overhead. Consequently, the use of this option should be reserved only for cases when the navigation to a specific page is highly probable if not absolutely certain.

The above suggestions are merely recommendations pertaining to the resource hints options in HTML5 as outlined by World Wide Web Consortium (W3C) [12]. Unfortunately, the actual implementation of the resource hints options in real-world browsers has neither been standardized nor mandated. As a result, there has been a significant variation in the number and actual implementation of different resource hints options by different browser types. (For more see [5], [6], [13]). Given that for the majority of Internet users Google Chrome happens to be the browser of choice [7], our discussion focuses on this particular browser type. Specifically, in Section 4, we present some of our experimental results pertaining to the behavior of Google Chrome when encountering different resource hints options in browsed pages.

2.2 Predictive Browsing in Chrome

As mentioned in the introduction, Chrome's predictive browsing is a feature intended to assist the user by speeding up the process of typing in a desired URL as well as retrieving its respective resources. The main mechanism behind this feature is the so-called Chrome prediction service, which runs independently on each Chrome instance. One of the main functions of this service is to maintain a database (i.e., a statistical record) of the past Web-page retrievals performed on the respective browser. This record, then, facilitates the calculation of the confidence/probability that one particular sequence of characters entered in the address bar of the given browser is going to actually segue into a request for one particular (previously requested) URL. (For an illustration see Figure 6.) This database can be accessed simply by typing in `chrome://predictors` in the browser's address bar.

☒ Filter zero confidences

Entries: 125

User Text	URL	Hit Count	Miss Count	Confidence
g	http://gmail.com/	594	186	0.7615384615384615
gi	http://githubarchive.org/	25	55	0.3125
gi	https://gist.github.com/	16	49	0.24615384615384617
gis	https://gist.github.com/	19	1	0.95
gist	https://gist.github.com/	19	1	0.95
githuba	http://githubarchive.org/	3	0	1
gm	http://gmail.com/	411	1	0.9975728155339806

Figure 6. Chrome predictors page [5]

It should be noted that each Chrome instance starts with no or very few entries in the predictor's database. As the number of times the user has used this browser increases, the database gets progressively more populated, resulting in more accurate predictions.

In the case of the browser whose predictors page is shown in Figure 6, there is a high confidence (0.761...) that whenever its user types 'g' in the address bar, the user is actually intending to type in (i.e., retrieve) gmail.com. For input 'gi', there is an almost equal likelihood that this might lead to the request for githubarchive.org or gist.github.com – with respective confidences of 0.3125 and 0.2461.... Finally, for input 'gm' it is almost certain, with confidence of 0.997..., that the user is aiming to type in (i.e., retrieve) gmail.com.

Now, a simple inspection of the source code for `chrome://predictors` (its respective CSS and JavaScript elements) reveals the following:

a) Entries with confidence levels greater than 0.8 get shaded green, entries with confidence levels between 0.5 and 0.8 get shaded yellow, and entries with confidence levels of below 0.5 are shaded grey.

b) In addition to impacting the color shading scheme, the confidence level of an entry also determines the set of actions performed by the browsers on the respective URL. In particular, whenever the user types in an entry

associated with the retrieval confidence of over 0.8 the respective URL/page is to be fully pre-rendered by the Chrome. If the retrieval confidence is between 0.5 and 0.8, the elements of the respective URL/page are (should be) pre-fetched. Finally, for the retrieval confidence of below 0.5, no preloading of the respective URL or its elements takes place.

3. EXPERIMENTAL SET-UP AND RESULTS

In order to gain a better understanding of how Google Chrome deals with different HTML5 resource hints options when encountering them in a browsed page, we have built an experimental client-servers framework as outlined in Figure 7. The ‘client’ in this framework is the latest version of Google Chrome (Chrome v.52) running on a laptop PC. The ‘server’ is set up on the Amazon Cloud (<http://ec2-54-186-72-100.us-west-2.compute.amazonaws.com>) and hosts a repository of test Web-pages. We’ve chosen to code the pages of this repository in php instead of plain html in order to be able to prevent their caching on the client side, as well as to be able to implement and examine the general impact of cookies on pages with embedded resource hints options.

The test pages of our framework are grouped into two sets. The pages of the first set are designed to be directly visible/accessible to the user, and each of them hides one particular resource hints option in its respective php/html code (pages A.php, B.php, C.php, D.php in Figure 7). The other set is comprised of pages referenced in the resource hints tags of the first set, and is not intended to be directly visible/accessible to the user (pages A_hidden.php,

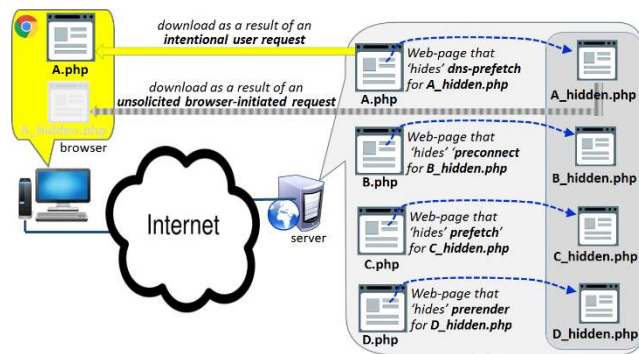


Figure 7. Experimental framework for evaluation of Chrome behavior when browsing pages with resource hints options

B_hidden.php, C_hidden.php, D_hidden.php in Figure 7). With this structure, if the pages of the second set - or their respective resources - ever get requested, that is a clear indication that the browser itself (not the user) has triggered those requests while processing the resource hints tags in the pages of the first set. (Note that, because of the way resource hints are intended to work as well as the way our framework is designed, requests for the pages of the second set not only get generated without the user’s direct knowledge and involvement, but the user also never gets to know when those resources actually arrive at their browser.)

In our experimentation, we first performed intentional requesting/retrieval of pages A.php to D.php (Figure 7) through the client - Chrome v.52 browser operating on a machine in our departmental network. Subsequently, we examined the collected artifacts pertaining to these requests both on the client and on the server side. The most significant of our observations are presented in Table 1, and can be summarized as follows:

Table 1. Artifacts collected on client and server side when resource hint options were found in a Web-page

resource hints option	browser-side artifacts				server-side artifacts
	effect on Chrome history	effect on Chrome cache	effect on Chrome DNS cache	effect on cookies	server side log
DNS-pre-fetch	<i>no effect</i>	<i>no effect</i>	<i>no effect</i>	<i>no effect</i>	<i>no GET request received at the server</i>
Pre-connect	<i>no effect</i>	<i>no effect</i>	<i>no effect</i>	<i>no effect</i>	<i>no GET request received at the server</i>
Pre-fetch	<i>no effect</i>	<i>Pre-fetched page/resource showed up in cache</i>	<i>showed up as a sub-resource of the calling website</i>	<i>cookies created</i>	<i>a GET request for pre-fetched resource/page received at the server (unless page/resource found in cache)</i>
Pre-render	<i>no effect</i>	<i>Pre-rendered page showed up in chrome cache</i>	<i>showed up as a standalone record (same as a user initiated visit)</i>	<i>cookies created</i>	<i>a GET request for pre-rendered page received at the server (unless page found in cache)</i>

1) The requesting of pages A.php and B.php (i.e., pages that contain DNS-pre-fetch and pre-connect resource hint options in their respective HTML5 code) did not leave any permanent artifacts related to A_hidden.php and B_hidden.php - either on the client or on the server side. Such a result could have been expected, as these two particular resource hints options do not ‘trigger’ application-level preloading of resources referenced in their <link> tags. Instead, DNS-pre-fetch and pre-connect facilitate only ‘lower level’ (DNS and TCP) domain-name resolution and connection set-up.

2) On the other hand, the requesting of pages C.php and D.php (i.e., pages that contain pre-fetch and pre-render resource hint options in their respective HTML5 code), did leave a number of artifact related to C_hidden.php and D_hidden.php on the client and on the server side. In particular:

2.a) On the client side, both (pre-fetched) C_hidden.php and (pre-rendered) D_hidden.php were not only retrieved but also ended up being stored in the browser cache. Furthermore, a cookie associated with each of the pages was created and placed in the browser’s cookie-database. Finally, a DNS record pertaining to both pages was stored in the browser’s DNS cache. All in all, the way the browser went about retrieving C_hidden.php and D_hidden.php was not much different from the way A.php to D.php were retrieved – even though the latter group of pages was explicitly requested by the user, while the user had no way of knowing that the former group of pages was ever requested/retrieved. (The only noticeable difference between the two groups is that the retrieval

of A.php to D.php was recorded in the browser history, which was not the case for C_hidden.php and D_hidden.php.)

- 2.b) On the server side, HTTP GET requests for both C_hidden.php and D_hidden.php appeared in the server logs. More importantly, these two requests looked identical to the requests for pages A.php to D.php, in terms of their (HTTP) content. In other words, based on what was recorded in the sever logs, was impossible to distinguish between the user's intentional requests (for A.php to D.php) and requests that were issued automatically by the browser without the user's knowledge and approval (for C_hidden.php and D_hidden.php).

Following the experimentation with the framework outlined in Figure 7, another experimental study was conducted where the Web objects referenced in A.php to D.php were pages hosted on another server. The observations concerning the recorded artifacts in this experiment were identical to the ones presented hereinabove (i.e., in Table 1).

Another important observation of our study is that in each Web-page with multiple pre-render tags/references, only one of these tags is executed at the time, while the respective (pre-rendered) Web-page ends up being placed in the browser's RAM.¹⁰ (The likely reason why Chrome and other browser do not allow simultaneous pre-rendering of multiple pages is to prevent potential overloading of the browser's RAM, which would degrade the overall browser performance.) On the other hand, there seem to be no limit on the number of pre-fetch tags/references that get executed in a Web-page. Once retrieved, each of the pre-fetched resources ends up being stored in the browser's cache.

4. PREBROWSING IMPLICATIONS ON USER PRIVACY AND FORENSICS

In this section, we present a few hypothetical (but very realistic) scenarios, in which pre-browsing mechanisms can be used as attack tools against one's reputation and privacy.

Scenario 1: Attack affecting victim's 'internal' reputation.

Imagine a situation where Trudy is a disgruntled employee working at a research company. Trudy holds a specific grudge towards Bob – a manager that she directly reports to. As a form of revenge against Bob, Trudy decides to format one of her upcoming reports as an HTML5 document. Inside this document, she 'hides' several dozens of resource hint tags – each pre-fetching¹¹ a highly inappropriate (e.g., adult-content or terrorism-related) Web-page. By means of JavaScript, Trudy also ensures that the execution of each pre-fetch tag occurs at a different point in time.

The 'reporting' day has come, and Bob opens the document that Trudy has referred him to. The (visible) content of the document seems very relevant, and Bob spends quite some time reading it. Clearly, while Bob is reading, his browser (in the background) retrieves/pre-fetches the inappropriate pages one-by-one, as illustrated in Figure 8. Bob, obviously, remains completely unaware that these downloads taking place.

At the same or later point in time, the company's Web-content firewall generates an alert pointing to Bob's machine (machine's IP) as the source of requests for inappropriate content. The company's authentication system verifies that the requests were generated while Bob was logged in and using the machine.

Now, depending on how severe the company's policy pertaining to inappropriate use of resources is, Bob could be subjected to a whole range of possible outcomes – from receiving a simple warning to facing serious disciplinary actions and possibly termination. The only way Bob could avoid these repercussions and clear his name is by providing aggregate browsing-related artifacts from his computer (spanning over a period of time before and after the actual incident) to relevant authorities. While adequate expert analysis of these artifacts would probably help 'put all the pieces of the puzzle together' and identify the actual cause of the inappropriate requests, the release of these artifacts could have potential negative implications on Bob's privacy, as they would contain Bob's entire

¹⁰ Our research has shown that, theoretically, it would be possible to have multiple pre-render tags, from one single Web-page, executed. Though, this would require that each of the pre-rendered Web-page comes with the auto-refresh functionality, and a relatively short auto-refresh interval.

¹¹ In this case, prefetching of a Web-page/URL would mean that its respective top-level resource (most often a base HTML file) is requested and retrieved.

browsing history within the given time period. Hence, Bob would ultimately have to weigh in the pros and cons of clearing his name vs. exposing the details of his browsing activity while at work. In either case, Bob is likely to experience unnecessary scrutiny for a period of time, with all the accompanying negative implications on his professional and personal life.

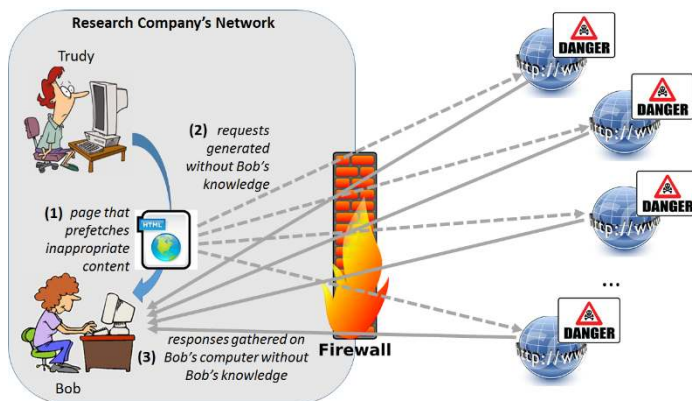


Figure 8. Reputation attack 1

Scenario 2: Attack affecting victim's 'external' reputation.

Now, imagine that in the previously depicted story, instead of tarnishing Bob's 'internal' reputation, Trudy decides to execute her revenge by affecting his 'outside' reputation. In particular, imagine that Trudy knows one particular Web-site that Bob likes to visit while at work (e.g., the Web-site of his bank or a specific news-agency Web-site). In that case, Trudy could hide a very large number of pre-fetch references targeting this particular Web-site (i.e., various pages/resources from this site) inside her 'malicious' Web-page, as illustrated in Figure 9.

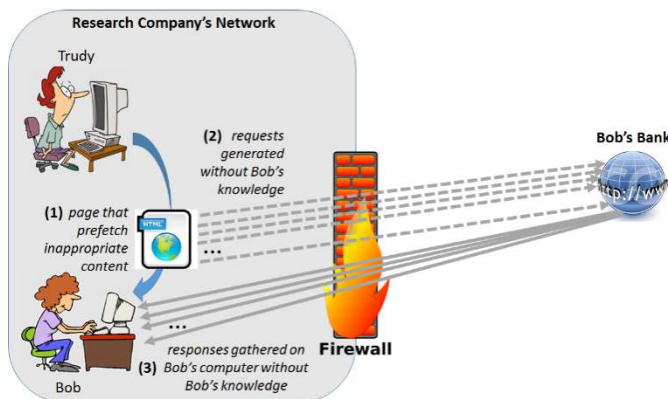


Figure 9. Reputation attack 2

Like many other similar organizations, Bob's bank is likely to be performing comprehensive intrusion detection monitoring of the incoming Web traffic, in order to spot and blacklist all misbehaving users. Given the avalanche of requests coming from Bob's machine (shown in Figure 9) is very reminiscent of a denial of service (DoS) attack, Bob's IP is likely to end up on the bank's blacklist, at least for a period of time. Clearly, during that period of time, even Bob's legitimate requests would be rejected (coming from the same IP), and Bob would be cut off from the online services of his bank.

Scenario 3: Attack affecting victim's privacy.

The third scenario that we are going to outline is less of an attack, and more of a set of unfortunate circumstances that could be imposed on the user's privacy.

Instead of providing desktops to its employees, Bob's company has implemented/encouraged 'Bring Your Own Device' (BOYD) policy. Hence, Bob decides to bring and use his personal laptop at work. However, for a while now Bob has not been particularly happy with his job, and has been intensively scanning for new job opportunities.

In fact, Bob goes to www.jobbank.com (a general job search site) and www.competitor.com/employment (the employment page of the main competitor to Bob's company) every day – to the point that these two URLs have gained the highest confidence scores (above 0.8) in the predictor database of Bob's Chrome browser. So, every time Bob starts typing www in the address bar of his Chrome browser, these URLs get suggested by Chrome's prediction service. Moreover, given the confidence score of these URLs of over 0.8, the actual pages associated with these URLs also get fully pre-rendered by his browser.

Now, recall, that the pre-rendering of a page means that the HTTP GET request for the given page, as well as the respective response, has traversed the communication path outlined in Figure 3. In Bob's case, this implies that the requests for the two Web-pages would be 'seen' by his organization's proxy – while Bob remains completely unaware of them. Depending on the organization's specific policies (or lack of thereof), it is possible that the information about these retrievals could end up in the hands of the upper management, which in turn could have implications on Bob's current and/or future prospects within this company.

5. CONCLUSIONS AND FUTURE WORK

The goal of this paper was to bring awareness to a slew of reputation and privacy related problems associated with the use of pre-browsing mechanism – specifically, HTML5 resource hints and Chrome's predictive browsing. We believe that the potential misuse of these mechanisms is serious enough to warrant a change in the way these mechanisms are implemented in today's browser (Web clients), as well as a change in the way Web traffic is logged on the server side. In particular, we believe that there should be mechanisms in place to allow for an easy separation between Web requests that were generated with the user's full intent and knowledge, and those that were not. Devising and implementing such mechanisms may not be a simple task, but keeping the 'status quo' does not seem acceptable both as it leaves each and every Web user vulnerable and exposed to a number of potential misuses. We plan to explore mechanisms that can either block such misuse or at least expose them in order to defend Web users. One such mechanism that we propose is to use the concept of browser obfuscation that is currently used in DuckDuckgo and Tor tools to be incorporated into existing browsers.

6. REFERENCES

1. Bowman, M., Debray, S. K., and Peterson, L. L. The importance of Web presence for small business. Simply Business, May 2013. DOI= <http://www.simplybusiness.co.uk/knowledge/articles/2013/05/why-web-presence-is-important/>. (1993).
2. Matt McGee. Google Instant Search: The Complete User's Guide. Search Engine Land, September 2010. DOI= <http://searchengineland.com/google-instant-complete-users-guide-50136>. (2010).
3. Danny Sullivan. How Google Instant's Autocomplete Suggestions Work. Search Engine Land, April 2011. DOI= <http://searchengineland.com/how-google-instant-autocomplete-suggestions-work-62592>. (2011).
4. Resource Hints. W3C Working Draft, 27 May 2016. DOI= <https://www.w3.org/TR/resource-hints/>. (2006).
5. Ilya Grigorik. High Performance Networking in Google Chrome. January 2013. DOI= <https://www.igvita.com/posa/high-performance-networking-in-google-chrome/>. (2013).
6. Ilya Grigorik. High Performance Browser Networking. O'Reilly, 2013.
7. StatCounter Global Stats. Top 5 Desktop, Tablet & Console Browsers from Aug 2015 to Aug 2016. DOI= <http://gs.statcounter.com/?PHPSESSID=oc1i9oue7por39rmhqq2eouoh0>. (2015).
8. C. Arthur. Why the default settings on your device should be right first time. theguardian, December 2013. DOI= <https://www.theguardian.com/technology/2013/dec/01/default-settings-change-phones-computers>. (2013).
9. M. Bichler. The Future of eMarkets: Multi-Dimensional Market Mechanisms. Cambridge University Press, (2001).
10. W³Tech Web Technology Surveys. Usage of Cookies for Websites. September 2016. DOI= <https://w3techs.com/technologies/details/ce-cookies/all/all> (2016).
11. February 2016 Web Server Survey. Netcraft, February 2016. DOI= <https://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html> (2016).
12. The World Wide Web Consortium W3C. DOI= <https://www.w3.org/>
13. B. Jackson. Resource Hints – What is Preload, Prefetch and Preconnect?. KeyCDN Blog. July 2016. DOI= <https://www.keycdn.com/blog/resource-hints/> (2016).
14. Jones Keith J. Forensic analysis of internet explorer activity files. Foundstone, http://www.foundstone.com/us/pdf/wp_index_dat.pdf; 2013.
15. Jones Keith j, Rohyt Blani. Web browser forensic. Security focus, <http://www.securityfocus.com/infocus/1827>; 2015a.

498 16.Jones Keith j, Rohyt Blani. Web browser forensic. Security focus, <http://www.securityfocus.com/infocus/1832>; 2015b.
499 17.Pereira Murilo Tito. Forensic analysis of the Firefox3 internet history and recovery of deleted SQLite records. Digital
500 Investigation; 2009:93e103. 5.
501