Web Browser History Detection as a Real-World Privacy Threat

Artur Janc[1] and Lukasz Olejnik[2]

artur@lingro.com,
lukasz.olejnik@man.poznan.pl

Abstract. Web browser history detection using CSS *visited* styles has long been dismissed as an issue of marginal impact. However, due to recent changes in Web usage patterns, coupled with browser performance improvements, the long-standing issue has now become a significant threat to the privacy of Internet users.

In this paper we analyze the impact of CSS-based history detection and demonstrate the feasibility of conducting practical attacks with minimal resources. We analyze Web browser behavior and detectability of content loaded via standard protocols and with various HTTP response codes. We develop an algorithm for efficient examination of large link sets and evaluate its performance in modern browsers. Compared to existing methods our approach is up to 6 times faster, and is able to detect up to 30,000 visited links per second.

We present a novel Web application capable of effectively detecting clients' browsing histories and discuss real-world results obtained from 271,576 Internet users. Our results indicate that at least 76% of Internet users are vulnerable to history detection, including over 94% of Google Chrome users; for a test of most popular Internet websites we were able to detect, on average, 62.6 (median 22) visited locations per client. We also demonstrate the potential to profile users based on social news stories they visited, and to detect private data such as zipcodes or search queries typed into online forms.

1 Introduction

Web browsers function as generic platforms for application delivery and provide various usability enhancements with implications for user privacy. One of the earliest such usability improvements was the ability to style links to Web pages visited by the user differently from unvisited links, introduced by the original version of the Cascading Style Sheets standard [1] and quickly adopted by all major Web browsers. This mechanism was soon demonstrated to allow malicious Web authors to detect links a client has visited and report them to the attacker [2].

Since then, a body of academic work has been published on this topic, describing history detection methods [3] and discussing the potential to detect visited websites to aid in phishing [4]. Several countermeasures against such attacks

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 215–231, 2010.
© Springer-Verlag Berlin Heidelberg 2010

were proposed, including client-side approaches through browser extensions [5] and server-side solutions on a per-application basis [6], but such methods have not been adopted by browser vendors or Web application developers. Simultaneously, several demonstration sites have been created to show the ability to detect known popular websites, including Web 2.0 applications [7].

More recently, CSS-based history detection started to become applied as a powerful component of privacy research, including work to determine the amount of user-specific information obtainable by ad networks [8] and as part of a scheme for deanonymizing social network users [9]. However, there has been a notable lack of work examining several crucial aspects of history detection, including the types of browser-supported protocols and resource types which can be detected, performance considerations, and the number of users affected by such attacks.

In this paper, we provide a detailed examination of CSS-based history detection techniques and their impact on the privacy of Internet users. We provide an overview of existing work, and discuss basic cross-browser implementations of history detection using JavaScript as well as a CSS-only technique. We evaluate the detectability of resources based on the browser-supported protocols used to retrieve them, analyze the effect of loading content in frames and iframes, as well as review the impact of HTTP redirects and other codes.

We demonstrate an optimized algorithm for detecting visited links and its JavaScript implementation. We provide detailed performance measurements of our technique and compare it to existing approaches. Our approach is up to 6 times faster than known methods, and allows for the examination of up to 30,000 links per second on modern hardware. We also provide the first performance analysis of the CSS-only history detection technique, demonstrating its value as an efficient, though often neglected, alternative to the scripting approach.

Based on our work on a real-world testing system [10], we provide an overview of the design of an efficient history detection application capable of providing categorized test of various website classes, and realizable with minimal resources. We discuss approaches for the selection of links to be detected, and outline our implemented solution based on *primary* links (as site entry points), *secondary* resources, and *enumeration elements*.

Finally, we analyze history detection results obtained from 271,576 users. We demonstrate that a large majority (76.1%) of Internet users are vulnerable to history detection attacks, including over 82% of Safari users and 94% of Google Chrome users. We analyze the average number of primary and secondary links found in a test of popular Internet locations; for vulnerable users our test found an average of 62.6 visited links (22 median). We also provide an overview of detected outgoing links from social news sites and discuss the potential of our system to gather especially privacy-sensitive data.

Our results indicate that properly prepared history detection attacks have significant malicious potential and can be directed against the vast majority of Internet users.

2 Background

The CSS *visited* pseudoclass has been applied to links visited by client browsers since the introduction of the CSS1 standard in 1996 [1]. The feature of applying different styles to “known” links quickly became accepted by users and was recommended by usability experts [11].

The ability to use the *visited* pseudoclass for detecting Web users’ browsing history was first reported to browser vendors as early as the year 2000 [2,12]. Since then, the technique has been independently rediscovered and disclosed several times [4], and has become widely known among Web browser developers and the security and Web standards communities. In fact, Section 5.11.2 of the CSS 2.1 standard [13], a W3C recommendation since 1998, discusses the potential for history detection using the *visited* pseudoclass, and explicitly allows conforming User Agents to omit this functionality for privacy reasons, without jeopardizing their compliance with the standard.

While initial discussions of CSS-based history detection were mostly conducted in on-line forums, the issue was also disclosed to the academic community and discussed in the work of Felten et al. in conjuction with cache-based history sniffing [3].

As a response, Jakobsson and Stamm discussed potential methods for implementing server-side per-application protection measures [14]; such techniques would have to be implemented by every Web-based application and are thus an extremely unlikely solution to the problem. A viable client-side solution was a proposed modification to the algorithm for deciding which links are to be considered visited as described in [5] and implemented in the SafeHistory extension [15] for Mozilla Firefox. Unfortunately, no such protection measures were implemented for other Web browsers[1], and the SafeHistory plugin is not available for more recent Firefox versions.

Other academic work in the area included a scheme for introducing voluntary privacy-oriented restrictions to the application of history detection[17]. Two more recent directions were applications of history detection techniques to determine the amount of user-specific information obtainable by ad networks [8] and as part of a scheme for deanonymizing social network users [9].

CSS-based history detection was also discussed as a potential threat to Web users’ privacy in several analyses of Web browser security [18,19].

Outside of the academic community, several demonstration sites were created to demonstrate specific aspects of browser history detection. Existing applications include a script to guess a visitor’s gender by combining the list of detected domains with demographic information from [20], a visual collage of visited Web 2.0 websites [7], and an entertaining detector of adult websites [21]. However, all known proof of concept sites focus on a single application, and do not explore the full potential of history detection as a tool to determine user-specific private information.

[1] Since writing the original draft of this work, we have become aware of ongoing efforts to mitigate history detection attacks in the Gecko and WebKit rendering engines [16].

3 Analysis

In order to fully evaluate the implications of CSS-based history detection, it is necessary to understand how and when Web browsers apply *visited* styles to links. In this section we analyze various browser behaviors related to visited links, describe an efficient algorithm for link detection and evaluate its performance in several major browsers[2].

3.1 Basic Implementation

CSS-based history detection works by allowing an attacker to determine if a particular URL has been visited by a client's browser through applying CSS styles distinguishing between visited and unvisited links. The entire state of the client's history cannot be directly retrieved; to glean history information, an attacker must supply the client with a list of URLs to check and infer which links exist in the client's history by examining the *computed* CSS values on the client-side. As noted in [12], there are two basic techniques for performing such detection.

The CSS-only method shown in Figure 1 allows an attacker's server to learn which URLs victim's browser considers to be visited by issuing HTTP requests for background images on elements linking to visited URLs. A similar, but less known technique is to use the *link* CSS pseudoclass, which only applies if the link specified as the element's href attribute has not been visited; the techniques are complementary.

```
<style>
#foo:visited {background: url(/?yes-foo);}
#bar:link {background: url(/?no-bar);}
</style>
<a id="foo" href="http://foo.org"></a>
<a id="bar" href="http://bar.biz"></a>
```

Fig. 1. Basic CSS Implementation

A similar technique can be performed on the client side with JavaScript, by dynamically querying the style of a link (<a>) element to detect if a particular CSS style has been applied, shown in Figure 2. Any valid CSS property can be used to differentiate between visited and unvisited links. The scripting approach allows for more flexibility on part of the attacker, as it enables fine-grained control over the execution of the hijacking code (e.g. allows resource-intensive

[2] Browser behavior and performance results were gathered with Internet Explorer 8.0, Mozilla Firefox 3.6, Safari 4, Chrome 4, and Opera 10.5 on Windows 7 using an Intel Core 2 Quad Q8200 CPU with 6GB of RAM.

tests to be run after a period of user inactivity) and can be easily obfuscated to avoid detection by inspecting the HTML source. It can also be modified to utilize less network resources than the CSS-only method, as discussed in Section 3.3. Both techniques can be executed transparently to the user and do not require any interaction other than navigating to a Web page.

```
<script>
var r1 = 'a {color:green;}';
var r2 = 'a:visited {color:red;}';

document.styleSheets[0].insertRule(r1, 0);
document.styleSheets[0].insertRule(r2, 1);

var a_el = document.createElement('a');
a_el.href = "http://foo.org";

var a_style = document.defaultView.getComputedStyle(a_el, "");

if (a_style.getPropertyValue("color") == 'red')
        // link was visited
</script>
```

Fig. 2. Basic JavaScript Implementation

3.2 Resource Detectability

The CSS history detection technique has historically been applied almost exclusively to detect domain-level resources (such as `http://example.org`), retrieved using the HTTP protocol. However, Web browsers apply the visited style to other kinds of links, including sub-domain resources such as images, stylesheets, scripts and URLs with local anchors, if they were visited directly by the user. In general, and with few exceptions, there exists a close correspondence between the URLs which appeared in the browser's *address bar* and those the browser considers to be visited. Thus, visited URLs within protocols other than HTTP, including `https`, `ftp`, and `file` can also be queried in all tested browsers, with the exception of Chrome which does not apply *visited* styles to `file://` links.

Because of the address bar rule outlined above, parameters in forms submitted with the HTTP POST request method cannot be detected, whereas parameters from forms submitted using HTTP GET are susceptible to detection. The URLs for resources downloaded indirectly, such as images embedded within an HTML document, are usually not marked as visited. However, one exception is the handling of frames and iframes in some browsers. A URL opened in a frame or iframe does not appear in the address bar, but the Firefox and Chrome browsers still consider it to be visited.

While all major browsers apply *visited* styles to valid resources (ones returning HTTP 200 status codes), variations exist for other response types. When encountering an HTTP redirect code (status 301 or 302) Firefox, Chrome and Opera mark both the redirecting URL and the new URL specified in the Location HTTP header as visited, whereas Safari saves only the original URL, and Internet Explorer exhibits seemingly random behavior. When performing a *meta redirect*, all browser except Internet Explorer consider both URLs to be visited; newer versions of Internet Explorer do not allow such redirects in the default configuration. When retrieving an invalid URL with a client or server error status (codes 4xx and 5xx), all browsers except Internet Explorer mark the link as visited. The handling of various types of HTTP responses is summarized in Table 1.

The ability to detect links visited by any user depends on the existence of those links in the browser's history store and is affected by history expiration policies. This default value for history preservation varies between browsers, with Firefox storing history for 90 days, Safari - 20 days, and IE - 20 days. Opera stores 1000 most recently visited URLs, whereas Chrome does not expire browsing history.

It is important to note that a potential adversary whose website is periodically visited by the user (or whose script is linked from such a site) can query the history state repeatedly on each visit, maintaining a server-side list of the user's detected links; such an approach would allow the attacker to aggregate browsing information, bypassing history expiration policies.

Table 1. Detectability for HTTP status codes and redirects

	IE	Firefox	Safari	Chrome	Opera
200	yes	yes	yes	yes	yes
301	random	both	original	both	both
302	random	both	original	both	both
meta redirect	n/a	both	both	both	both
404	no	yes	yes	yes	yes
500	no	yes	yes	yes	yes

3.3 Performance

CSS-based history detection is a viable technique for various privacy-related attacks because of its simplicity and the ability to quickly check for a large number of visited resources. In order to fully understand the implications of CSS-based history detection attacks, it is thus crucial to learn about its performance characteristics using optimized scripts for each browsing environment.

Optimizing JavaScript Detection. To date, little effort has been put into the analysis of efficient implementations of JavaScript-based history detection. Several existing implementations use DOM `a` elements in a static HTML document to hold URLs which are later inspected to determine if the CSS visited rule

applied to the corresponding URL, an approach significantly slower than a fully-dynamic technique. Additionally, due to browser inconsistencies in their internal representations of computed CSS values (e.g. the color red can be internally represented as "red", "#ff0000", "#f00", or "rgb(255, 0, 0)") most detection scripts try to achieve interoperability by checking for a match among multiple of the listed values, even if the client's browser consistently uses one representation. Another difference affecting only certain browsers is that an `a` element must be appended to an existing descendant of the document node in order for the style to be recomputed, increasing script execution time.

For our detection code, we took the approach of creating an optimized technique for each major browser and falling back to a slower general detection method for all other browsers. We then compared the execution time of the optimized test with the general method for each major browser. The differences in execution times are shown in Figure 3.

For each browser the implementation varies slightly, depending on the way CSS properties are represented internally and the available DOM mechanisms to detect element styles. The general detection algorithm for lists of links is as follows:

1. Initialize CSS styles and store URLs to check in a JavaScript array.
2. Detect browser version and choose appropriate detection function.
3. Invoke chosen detection function on URL array.
   - Create <**a**> element and other required elements.
   - For each URL in array:
     - Set <**a**> element href attribute to URL.
     - (for some browsers) Append element to DOM or recompute styles.
     - If computed style matches visited style, add URL to "visited" array.
4. Send contents of visited array to server or store on the client-side.

Our approach has the advantage of avoiding a function call for each check, reusing DOM elements where possible, and is more amenable to optimization by JavaScript engines due to a tight inner loop. Compared to a naive detection approach using static <**a**> elements in the HTML source and less-optimized style matching, our technique is between 1.8 and 6 times faster depending on the browser.

CSS Performance. The CSS-only detection technique is a valuable alternative to the scripting approach, as it allows to test clients with JavaScript disabled or ones with security-enhancing plug-ins such as NoScript. Our results, provided in Figure 4, show that CSS-based detection can perform on par with the scripting approach, allowing an attacker to test for over 25,000 visited links per second for small data sets of 50,000 links and fewer. An important drawback, however, is that CSS-based detection requires <**a**> elements with appropriate *href* attributes to be included in the static HTML source, increasing the page size and required bandwidth. Additionally, for larger link sets (HTML pages with over

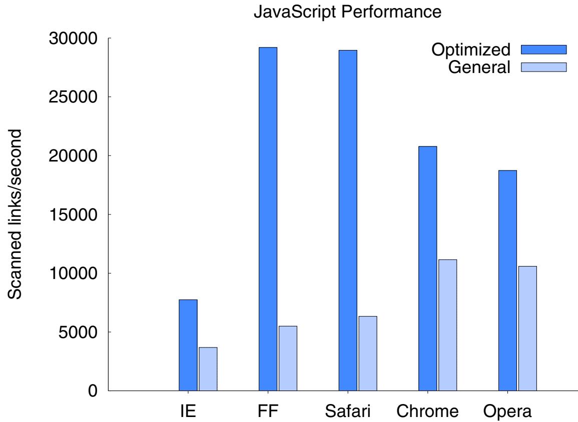

Fig. 3. JavaScript detection performance for different browsers. The general approach can be clearly seen as much slower.

50,000 elements), detection performance (and overall browser performance) decreases quickly with the increasing number of DOM elements included in the page[3].

Network Considerations. While client-side detection efficiency is of the most importance, we observe that the overall goal of detecting visited URLs in the client's browsing history can require significant network resources. Since many browsers on modern hardware are able to check tens of thousands of links per second, the bandwidth necessary to sustain constant checking speed becomes non-trivial.

In our test data set, the median URL lengths are 24 bytes for primary links (hostnames), and 60 bytes for secondary links (resources within each site). The overhead of including a URL in a JavaScript script in our implementation was 3 bytes (for quoting and separating array elements). For CSS, the average size overhead was 80 bytes due to the necessity of adding HTML markup and static CSS styles. In our tests, transmitting 30,000 thousand URLs required approximately 1650 kB (170 kB with gzip compression) for JavaScript, and 3552 kB (337kB with gzip compression) for CSS tests.

For an average broadband user, available bandwidth could potentially be a limiting factor, especially for owners of modern systems which can execute the detection code faster. To decrease the required bandwidth, transmitted links can omit common patterns (e.g. `http://` or `http://www.`); enumerating resources within a single domain can also significantly reduce the required network bandwidth by only transmitting the variable URL component.

[3] Test pages with more than 50 thousand elements caused errors and did not load in Internet Explorer.

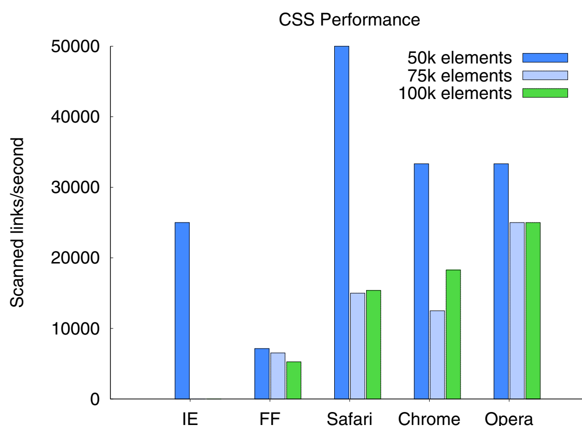

Fig. 4. CSS detection performance. Due to the limitations of Internet Explorer, only data for 50 thousand links is shown.

4 Methodology

A core goal of our work was to build a functional system to demonstrate the possible privacy risks associated with browser history detection, including the development of categorized tests detecting various classes of online resources. Our testing system was designed to maximize the number of URLs retrieved from each visitor's history and to present visitors with a visual representation of what can be inferred about their browsing habits.

4.1 System Overview

Our testing application was divided into multiple test categories, each of which contained several history detection tests. Test categories included:

– General tests of popular websites selected from Web rankings [22],
– On-line news and social news sites along with posted story links,
– A final category of miscellaneous tests (including a zipcode detection test and a check of performed search engine queries).

The default test which executed when a user visited the site homepage was the "top5k" test, checking for 6,417 most popular Internet locations. Selected tests are listed in Table 2.

When a user visited a test page, she was presented with a short test description, along with a list of *primary* links to check. When the page loaded, the browser automatically performed checks of all links in the list, continuously updating a progress bar to inform the user about the test status. When all links were checked, the browser submitted the results to the server using an AJAX request, and received in response the thumbnail images and descriptions for all

websites for which primary links were found, as well as a list of *secondary* links for each such website. The browser then checked all links in the secondary list and submitted the results to the server. The server's final reply contained an overview of the data found in the user's history, along with a detailed list of all primary and secondary links found.

For some tests, the set of secondary links was accompanied by a list of *enumeration elements* such as usernames on a visited social news site (Digg, Reddit or Slashdot), popular search engine queries for the search query test, or US zipcodes for the zip code detector test. Enumeration elements were appended to one or more base URLs supplied by the server (of the form `http://reddit.com/user/username`, with *username* as an enumeration element) and were checked similarly to primary and secondary links. This mechanism added a semantic component to the test by informing the server about the type of the link found in the user's history (e.g. username or search term), as contrasted with a "generic" link. It also helped the system conserve network bandwidth, by omitting common URL prefixes for similar resources.

If a user visited any test page with JavaScript disabled, the server automatically recognized that fact and redirected the client to a separate test page which utilized the CSS-only method described in Section 3.1. The CSS-only test required more network resources, but tested for the same primary and secondary links as the regular test and presented results in the same manner. An overview of differences between results gathered from clients with and without JavaScript is provided in Table 3.

4.2 Link Selection

The selection of URLs to check for in each client's history is of paramount importance in any project utilizing CSS-based history detection, as it determines how much browsing data can be gathered. However, if too much data is transferred to the user, both the page load and test run times might increase to the point that the user will leave the page without completing the test. Large data sets also limit the number of concurrent client a testing server can support due to server-side network and computational limitations. In our system we tackled this problem by both splitting tests into domain-specific categories, and dividing our tests into two phases for checking *primary* and *secondary* links.

Primary Links. For each test we gathered primary links representing domains of websites which contained resources of interest for the particular test. For the general test category we used popular Web analytics services including Alexa [22], Quantcast [20] and Bloglines [23] to identify the most popular Internet locations.

We retrieved the HTML source for each primary link and if any HTTP redirects occured, we kept track of the new URLs and added them as alternate links for each URL (for example if `http://example.org` redirected to `http://example.org/home.asp` both URLs would be stored). We also performed basic

unifications if two primary links pointed to slightly different domains but appeared to be the same entity (such as `http://example.org` and `http://www.example.org`).

A total of 72,134 primary links were added to our system as shown in Table 2. To each primary link we added metadata, including the website title and a human-readable comment describing the nature of the site, if available. Primary links served as a starting point for resource detection in each test—if a primary link (or one of its alternate forms) was detected in the client's history, secondary links associated with that primary link were sent to the client and checked.

Table 2. Number of links to be scanned per test is shown

	Primary links	Secondary links
top5k	6417	1416709
top20k	23797	4054165
All	72134	8598055

Secondary Links. Browser history detection has the potential for detecting a variety of Web-based resources in addition to just the hostname or domain name of a service. In our tests, for each primary link we gathered a large number of *secondary* links for resources (subpages, forms, directly accessible images, etc.) within the domain represented by the primary link. The resources were gathered using several techniques to maximize the coverage of the most popular resources within each site:

1. Search engine results. We utilized the Yahoo! BOSS [24] search engine API and queried for resources within the domain of the primary link, taking advantage of the fact that search engine results are sorted by relevance so that the top results returned correspond to the most often visited pages. For most primary links, we requested 100 results, but for the most popular Internet locations (sites in the Alexa 500 list) we retrieved 500 results.
2. HTML inspection. We retrieved the HTML source for each primary link and made a list of absolute links to resources within the domain of the primary link. The number of secondary links gathered using this method varied depending on the structure of each site.
3. Automatic generation. For some websites with known URL schemes we generated secondary links from list pages containing article or website section names; this behavior allowed us to quickly generate links for websites such as Craigslist and Wikileaks.

We then aggregated the secondary links retrieved with each method, removing duplicates or dubious URLs (including ones with unique identifiers which would be unlikely to be found in any user's history) and added metadata such as link descriptions where available.

For news site tests we also gathered links from the RSS feeds of 80 most popular news sites, updated every two hours[4]. Each RSS feed was tied to a primary link (e.g. the `http://rss.cnn.com/rss/cnn_topstories.rss` was associated with the `http://cnn.com` primary link). Due to the high volume of links in some RSS feeds, several news sites had tens of thousands of secondary links.

Resource Enumeration. In addition to secondary links, some primary links were also associated with *enumeration elements*, corresponding to site-specific resources which might exist in the browser's cache, such as usernames on social news sites, popular search engine queries, or zipcodes typed into online forms. To demonstrate the possibility of deanonymizing users of social news sites we gathered lists of active users on those sites by screen scraping for usernames of link submitters and comment authors. Enumeration elements were also useful for tests where similar resources might be visited by the user on several sites – in our search engine query test, the URLs corresponding to searches for popular phrases were checked on several major search engines without needing to transmit individual links multiple times.

4.3 Processing Results

For each visiting client, our testing system recorded information about the links found in the client's history, as well as metadata including test type, time of execution, the User Agent header, and whether the client had JavaScript enabled. Detected primary links were logged immediately after the client submitted first stage results and queried the server for secondary links. After all secondary links were checked, the second stage of detection data was submitted to the test server. All detected information was immediately displayed to the user.

For large-scale history detection systems, the amount of gathered data might be affected by server-side resource limits such as bandwidth and processing power. Our application was deployed on a single virtual server in a shared VM environment [25] using a basic $20/month plan, which affected the amount of information we could gather and process. Organizations with more resources would be able to perform more extensive history detection tests, posing a more serious threat to user privacy.

5 Results

The testing application based on this work was put into operation in early September 2009 and is currently available at [10]. Results analyzed here span the period of September 2009 to February 2010 and encompass data gathered from 271,576 users who executed a total of 703,895 tests. The default top5k

[4] Due to high interest in our testing application and associated resource constraints, we were forced to disable automatic updating of RSS feeds for parts of the duration of our experiment.

test, checking for 6,417 most popular Internet locations and 1,416,709 secondary URLs within those properties was executed by 243,068 users[5].

5.1 General Results

To assess the overall impact of CSS-based history detection, it is important to determine the number of users whose browser configuration makes them vulnerable to the attack. Table 3 summarizes the number of users for whom the top5k test found at least one link, and who are therefore vulnerable. We found that we could inspect browsing history in the vast majority of cases (76.1% connecting clients), indicating that millions of Internet users are at risk. A somewhat smaller number of users with found results for the All test might be attributed to the fact that users who recently cleared their browsing history or used private browsing modes executed the most extensive test to determine if they are at any risk.

Table 3. Aggregate results for popular tests for JavaScript and CSS-only techniques

Test	Tests Ran		Found Primary		Primary/user (avg)		Secondary/user (avg)	
	JS	CSS	JS	CSS	JS	CSS	JS	CSS
top5k	206437	8165	76.1%	76.9%	12.7	9.8	49.9	34.6
top20k	31151	1263	75.4%	87.3%	13.6	15.1	48.1	51.0
All	32158	1325	69.7%	80.6%	15.3	20.0	49.1	61.2

An analysis of relative differences in susceptibility to history detection based on the user agent is shown in Table 4. For all browsers, the number of clients who were found vulnerable was above 70%. Browsers such as Safari and Chrome reported higher rates of susceptible clients (82% and 94% average), indicating that history detection can affect a significant number of Internet *power users*.

For users with at least one detected link tested with the JavaScript technique we detected an average of 12.7 websites (8 median) in the top5k list, as well as 49.9 (17 median) secondary resources. Users who executed the more-extensive JavaScript top20k test, were detected to have visited an average of 13.6 (7) pages with 48.2 (15) secondary resources. Similar results were returned for clients who executed the most elaborate `all` test, with 15.3 (7) primary links and 49.1 (14) secondary links. The distribution of top5k results for JavaScript-enabled browsers is shown in Figure 5. An important observation is that for a significant number of users (9.5%) our tests found more than 30 visited primary links; such clients are more vulnerable to deanonymization attacks and enumeration of user-specific preferences.

[5] Our testing system was featured on several social news sites and high-readership blogs, which increased the number of users who visited our website and helped in the overall data acquisition.

Table 4. Percentage of clients with detected links by User Agent

Test	IE		Firefox		Safari		Chrome		Opera	
	JS	CSS	JS	CSS	JS	CSS	JS	CSS	JS	CSS
top5k	73	92	75	77	83	79	93	100	70	82
top20k	81	95	69	86	89	97	90	100	88	95
All	78	97	62	79	85	89	87	98	85	83

Due to the fact that our testing site transparently reverted to CSS-only tests for clients with scripting disabled, we are also able to measure the relative differences in data gathered from clients with JavaScript disabled or unavailable. A total of 8,165 such clients executed the top5k test; results were found for 76.9% of clients, with a median of 5 visited primary URLs and 9 secondary URLs. Results for the top20k test executed in CSS yielded results similar to JavaScript clients, with 15.1 (8) websites and 51.0 (13) secondary links.

Interestingly, it seems that for certain tests, users without JavaScript appear more vulnerable due to a higher number of clients with at least one found link, and more detected links per client. This result should be an important consideration for organizations which decide to disable scripting for their employees for security reasons, as it demonstrates that such an approach does not make them any more secure against history detection attacks and associated privacy loss.

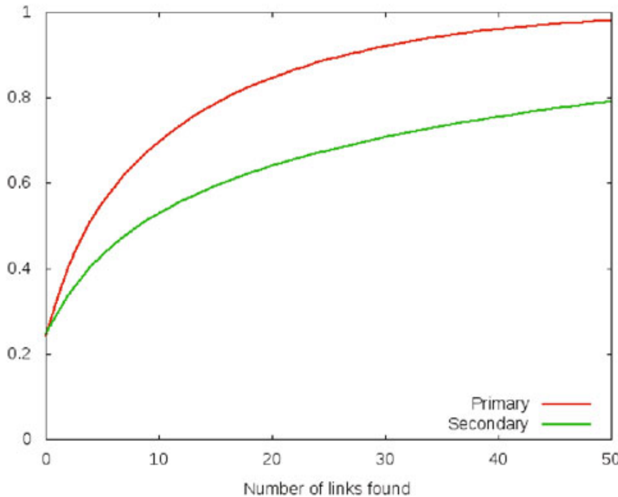

Fig. 5. Cumulative distribution of top5k primary and secondary links

5.2 Social News Site Links

An important overall part of our test system were tests of visited links from social news sites. We investigated three popular social news sites: Digg, Reddit and Slashdot. For each site, in addition to secondary links representing popular pages within that website, we also gathered all links to external destinations from the site's main RSS feed. We also checked for visits to the profile pages of active users within each site using the enumeration strategy outlined in Section 4.2.

We found that for users whose browsing history contained the link of the tested social news site, we could, in a significant majority of cases, detect resources linked from that site. History detection techniques could be used to measure user engagement on social news sites by comparing the average number of visited news stories; such analysis can be done both for individual users, and on aggregate, as a tool to compare social news site popularity. Additionally, for 2.4% of Reddit users we found that they visited the profile of at least one user of their social news site. Such data demonstrates that it is possible to perform large-scale information gathering about the existence of relationships between social news site users, and potentially deanonymize users who visit their own profile pages.

Table 5. Average and median numbers of found secondary links from social news sites

	Average secondary	Median secondary
Digg	51.8	7
Reddit	163.3	26
Slashdot	15.2	3

It is important to note that the specified architecture can potentially be used to determine user-specific preferences. Inspecting detected secondary links can allow a determined attacker to not only evaluate the relationship of a user with a particular news site, but also make guesses about the type of content of interest to the particular user.

5.3 Uncovering Private Information

For most history tests our approach was to show users the breadth of information about websites they visit which can be gleaned for their browsing history. However, we also created several tests which used the resource enumeration approach to detect common user inputs to popular web forms.

The zipcode test detected if the user typed in a valid US zipcode into a form on sites requiring zipcode information (there are several sites which ask the user to provide a zipcode to get information about local weather or movie showtimes). Our analysis shows that using this technique we could detect the US zipcode for as many as 9.2% users executing this test. As our test only covered several hand-picked websites, it is conceivable that with a larger selection of websites

requiring zip codes, the attack could be easily improved to yield a higher success rate.

In a similar test of queries typed into the Web forms of two popular search engines (Google and Bing) we found that it is feasible to detect some user inputs. While the number of users for whom search terms were detected was small (about 0.2% of users), the set of terms our test queried for was small (less than 10,000 phrases); we believe that in certain targeted attack scenarios it is possible to perform more comprehensive search term detection.

While limited in scope due to resource limitations, our results indicate that history detection can be practically used to uncover private, user-supplied information from certain Web forms for a considerable number of Internet users and can lead to targeted attacks against the users of particular websites.

6 Conclusions

This paper describes novel work on analyzing CSS-based history detection techniques and their impact on Internet users. History detection is a consequence of an established and ubiquitous W3C standard and has become a common tool employed in privacy research; as such, it has important implications for the privacy of Internet users. Full understanding of the implementation, performance, and browser handling of history detection methods is thus of high importance to the security community.

We described a basic cross-browser implementation of history detection in both CSS and JavaScript and analyzed Web browser behavior for content returned with various HTTP response codes and as frames or iframes. We provided an algorithm for efficient examination of large link sets and evaluated its performance in modern browsers. Compared to existing methods our approach is up to 6 times faster, and is able to detect up to 30,000 links per second in recent browsers on modern consumer-grade hardware. We also provided and analyzed results from our existing testing system, gathered from total number 271,576 of users. Our results indicate that at least 76% of Internet users are vulnerable to history detection; for a simple test of the most popular websites, we found, on average 62.6 visited URLs.

Our final contribution is the pioneering the data acquisition of history-based user preferences. Our analysis not only shows that it is feasible to recover such data, but, provided that it's large-scale enough, enables enumeration of privacy-relevant resources from users' browsing history. To our knowledge, this was the first such attempt. Our results prove that CSS-based history detection does work in practice on a large scale, can be realized with minimal resources, and is of great practical significance.

Acknowledgements. L.O. gratefully acknowledges financial support for this work from the European Organization for Nuclear Research (CERN) and, in particular, N. Neufeld for the help, support and fruitful discussions.

References

1. W3C: Cascading style sheets, level 1, http://www.w3.org/TR/REC-CSS1/
2. Bugzilla: Bug 57351 - css on a: visited can load an image and/or reveal if visitor been to a site (2000), https://bugzilla.mozilla.org/show_bug.cgi?id=57531
3. Felten, E.W., Schneider, M.A.: Timing attacks on web privacy. In: CCS 2000: Proceedings of the 7th ACM Conference on Computer and Communications Security, pp. 25–32. ACM, New York (2000)
4. Jagatic, T.N., Johnson, N.A., Jakobsson, M., Menczer, F.: Social phishing. ACM Commun. 50(10), 94–100 (2007)
5. Jackson, C., Bortz, A., Boneh, D., Mitchell, J.C.: Protecting browser state from web privacy attacks. In: WWW 2006: Proceedings of the 15th International Conference on World Wide Web, pp. 737–744. ACM, New York (2006)
6. Jakobsson, M., Stamm, S.: Web camouage: Protecting your clients from browsersning attacks. IEEE Security and Privacy 5, 16–24 (2007)
7. Webcollage: Web 2.0 collage, http://www.webcollage.com/
8. Wills, C.E., Zeljkovic, M.: A personalized approach to web privacy–awareness, attitudes and actions. Technical Report WPI-CS-TR-10-07, Computer Science Department, Worcester Polytechnic Institute (2010), http://www.cs.wpi.edu/~cew/papers/whattheyknow.pdf
9. Wondracek, G., Holz, T., Kirda, E., Kruegel, C.: A practical attack to de-anonymize social network users, IEEE security and privacy. In: IEEE Security and Privacy, Oakland, CA, USA (2010)
10. Janc, A., Olejnik, L.: What the internet knows about you, http://www.wtikay.com/
11. Nielsen, J.: Change the color of visited links, http://www.useit.com/alertbox/20040503.html
12. Bugzilla: Bug 147777 - :visited support allows queries into global history (2002), https://bugzilla.mozilla.org/show_bug.cgi?id=147777
13. W3C: Cascading style sheets level 2 revision 1 (css 2.1) speci_cation, selectors, http://www.w3.org/TR/CSS2/selector.html#link-pseudo-classes
14. Jakobsson, M., Stamm, S.: Invasive browser sniffing and countermeasures. In: WWW 2006: Proceedings of the 15th International Conference on World Wide Web, pp. 523–532. ACM, New York (2006)
15. Jackson, C., Andrew Bortz, D.B.J.M.: Stanford safehistory, http://safehistory.com/
16. Baron, L.D.: Preventing attacks on a user's history through css : visited selectors (2010), http://dbaron.org/mozilla/visited-privacy
17. Jakobsson, M., Juels, A., Ratkiewicz, J.: Privacy-preserving history mining for web browsers. In: Web 2.0 Security and Privacy (2008)
18. Zalewski, M.: Browser security handbook, part 2 (2009), http://code.google.com/p/browsersec/wiki/Part2
19. König, F.: The art of wwwar: Web browsers as universal platforms for attacks on privacy, network security and rbitrary targets. Technical report (2008)
20. Quantcast: Quantcast, http://www.quantcast.com/
21. Anonymous: Did you watch porn, http://didyouwatchporn.com
22. Alexa: Alexa 500, http://alexa.com
23. Bloglines: Bloglines top feeds, http://www.bloglines.com/topblogs
24. Yahoo!: Yahoo! boss, http://developer.yahoo.com/search/boss/
25. Linode: Linode vps hosting, http://linode.com