Feasibility and Real-World Implications of Web Browser History Detection

Artur Janc
artur@lingro.com

Lukasz Olejnik
lukasz.olejnik@man.poznan.pl

ABSTRACT

Browser history detection through the Cascading Style Sheets *visited* pseudoclass has long been known to the academic security community and browser vendors, but has been largely dismissed as an issue of marginal impact.

In this paper we present several crucial real-world considerations of CSS-based history detection to assess the feasibility of conducting such attacks in the wild. We analyze Web browser behavior and detectability of content returned via various protocols and 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 as many as 30,000 links per second in recent browsers on modern consumer-grade hardware.

We present a web-based system capable of effectively detecting clients' browsing histories and categorizing detected information. We analyze 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; for a test of most popular Internet websites we were able to detect, on average, 62 visited locations. We also demonstrate the potential for detecting private data such as zipcodes or search queries typed into online forms. Our results confirm the feasibility of conducting attacks on user privacy using CSS-based history detection and demonstrate that such attacks are realizable with minimal resources.

1. INTRODUCTION

Web browsers function as generic platforms for application delivery and provide various usability enhancements and performance optimizations, many of which have 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 CSS standard[2] and quickly adopted by all major Web browsers. This mechanism was quickly demonstrated to allow malicious Web authors to detect which links a client had[1].

Since then, a body of academic work was created on the topic, describing history detection methods[13] and discussing the potential to detect visited websites to aid in phishing[16, 21]. Countermeasures against such attacks were proposed, including client-side approaches through browser extensions[15] and server-side solutions on a per-application basis[17], 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]. Initial academic work and all known demonstrations focused on gathering information about the domain names of known sites and potentially applying it for inferring user-related information.

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[9] and as part of a scheme for deanonymizing social network users[14].

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. Perhaps because of the lack of such large-scale studies no Web browser vendor has yet implemented any kind of protection against history detection attacks[1].

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 response codes.

We also demonstrate an optimized algorithm for detecting visited links and its JavaScript implementation. We provide detailed performance measurements and compare it to known methods. Our approach is, up to six 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 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[8], 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*.

[1]Since writing the original draft of this article, we have learned of plans to mitigate against such attacks in an upcoming release of the Mozilla Firefox browser[12].

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. We analyze the average number of primary and secondary links found for a basic test, and provide examples of privacy-sensitive data gathered by our system.

Our results indicate that properly prepared history detection attacks have significant malicious potential and can be targeted 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 an early stage of the Web, in 1996[2]. The feature of applying different styles to "known" links quickly became accepted by users and was recommended by various usability experts[4].

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[11, 1]. Since then, the technique has been independently rediscovered and disclosed several times[16], and has become widely known among Web browser developers and the security and Web standards communities. In fact, Chapter 5.11.2 of the CSS 2.1 standard[3], 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 Felden et al. in conjuction with cache-based history sniffing[13].

As a response, Jakobsson and Stamm discussed potential methods for implementing server-side per-application protection measures[17]; 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 [15] and implemented in the SafeHistory extension[6] for Mozilla Firefox. Unfortunately, no such protection measures were implemented for other Web browsers, 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[20]. Two more recent directions were applications of history detection techniques to determine the amount of user-specific information obtainable by ad networks [9] and as part of a scheme for deanonymizing social network users[14].

CSS-based history detection was also discussed as a potential threat to Web users' privacy in several analyses of Web browser security[23][18].

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 [5] and a visual collage of visited Web 2.0 websites[7]. All such known demonstrations focus on detecting domains only, and do not attempt to discover any intra-domain resources or attach semantic information to detected links.

```
<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>
```

Figure 1: Basic CSS Implementation.

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 styles to visited 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 [11], there are two basic techniques for performing such detection.

The CSS-only method, as 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.

This basic CSS functionality can also be accessed within JavaScript by dynamically querying the style of a link (`<a>`) element to detect if a particular CSS style has been applied, as 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 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.

Both the CSS styling of visited links, and the JavaScript `getComputedStyle` function have many legitimate uses and are relied upon by Web developers. The ability to apply different styles to visited links serves as a powerful visual aid in website navigation, and was one of the requirements during

[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.

```
<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>
```

Figure 2: Basic JavaScript Implementation.

Table 1: Detectable Protocols by Browser.

	IE	Firefox	Safari	Chrome	Opera
http	✓	✓	✓	✓	✓
https	✓	✓	✓	✓	✓
ftp	✓	✓	✓	✓	✓
file	✓	✓	✓		✓

the development of the original CSS standard[19]. The `getComputedStyle` method is often used for determining the size and dynamic resizing of HTML containers and is crucial correctly laying out a large number of websites. Therefore, any changes to those mechanisms are likely to result in problems for a portion of the Web[12], or potentially introduce accessibility issues by violating W3C accessibility guidelines[22]. Browser vendors attempting to mitigate against history detection attacks will thus need to justify such changes by explaining the privacy issues associated with styling visited links to both users and Web developers.

3.2 Resource Detectability

CSS history detection has historically been applied almost exclusively to detect domain-level resources (such as http://example.org), retrieved using the HTTP protocol (a notable recent exception is [14] which is based on the detailed enumeration of visited resources within each domain to deanonymize social network users). However, Web browsers apply the *visited* style to various kinds of visited 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 most browsers, as shown in Table 1.

Because of the address bar rule outlined above, parameters in forms submitted with the HTTP POST request method cannot be detected using the described technique, whereas parameters from forms submitted using HTTP GET can. 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 some browsers, including Firefox and Chrome, still consider it to be visited, as shown in Table 2.

Table 2: Detectability of frame and iframe URLs.

	IE	Firefox	Safari	Chrome	Opera
frames		✓		✓	
iframes		✓		✓	

Table 3: Detectability by HTTP status codes.

	IE	Firefox	Safari	Chrome	Opera
200	✓	✓	✓	✓	✓
301	n/a	both	original	both	both
302	n/a	both	original	both	both
4xx		✓	✓	✓	✓
5xx		✓	✓	✓	✓
meta	n/a	✓	✓	✓	✓

While all major browsers behave as expected when downloading 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 retrieving an invalid URL (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 3.

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, 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 included on such a site) can query the history state repeatedly on each visit, maintaining a server-side list of detected links.

3.3 Performance

CSS-based browsing history detection has become a popular technique for various privacy-related attacks because of its simplicity and the ability to quickly check for a large number of visited resources. Compared to other easily realizable attacks, such as the ones described in [13], the main advantage is that this approach is non-destructive (does not alter the browser's history state), and its speed. 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.

3.3.1 Optimizing JavaScript detection

To date, little effort has been put into researching 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 entirety of the general detection algorithm is as follows:

1. Initialize visited and unvisited styles, and URLs to check in a JavaScript array.
2. Detect browser and choose 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 up to 6 times faster, as depicted in Figure 3.

3.3.2 CSS Performance

The CSS-only detection technique, while less flexible and difficult to optimize, is also a valuable alternative to the scripting detection approach, as it allows to test clients with JavaScript disabled or ones with security-enhancing plugins such as NoScript. Our results, provided in Figure 4, show that CSS-based detection can perform on par with the scripting approach 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,

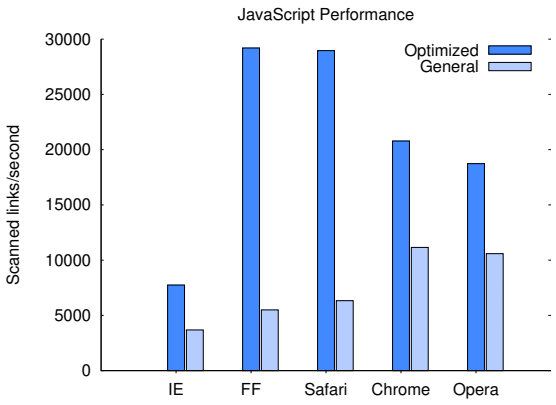

Figure 3: JavaScript detection performance

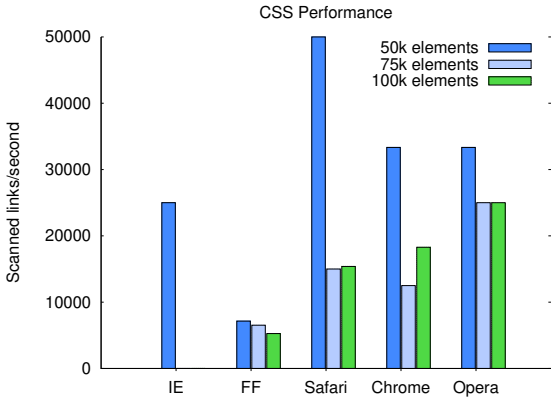

Figure 4: CSS detection performance.

for more sizable link sets (HTML pages with over 50,000 elements), detection performance (and overall browser performance) decreases quickly due to a large number of DOM elements included in the page.

3.3.3 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 length for a hostname is 24 bytes, and for an internal (secondary) resource is 60 bytes. 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 enumerated component.

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. As such, our testing system was designed to maximize the number of potentially private 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 web application was divided into multiple test categories, each of which contained several history detection tests. Test categories included:

- General tests of popular websites,
- Websites with sensitive content (e.g. Wikileaks article pages and visited adult websites),
- On-line news and social news websites along with story links posted on such sites, and
- A final category of miscellaneous tests (including a zipcode detection test and a check of 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 4.

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 final server 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 (such as `http://reddit.com/user/`) 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 links.

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 5.

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.

4.2.1 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, Quantcast and Bloglines to identify the most popular Internet locations. For other tests, such as the social network, bank or government/military site tests, we compiled extensive lists of sites within each category by combining several available online lists and manually verifying their accuracy. There was a many-to-many mapping between primary links and tests; each test was composed of thousands of primary links, and some links were checked in multiple tests - a popular on-line store such as Amazon.com was checked in the popular site tests (`top5k`, `top20k`) as well as the domain-specific online store test.

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 4. 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.

4.2.2 Secondary Links

As discussed in Section 3, 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 ser-

Table 4: Test link counts.

	Primary links	Secondary links
top5k	6417	1416709
top20k	23797	4054165
All	72134	8598055
Adult	6732	331051

vice. 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[10] search engine API and queried for all resources within the domain of the primary link. For each such link we gathered between 100 and 500 resources.

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 organization 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.

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[3]. 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 contained tens of thousands of secondary links.

4.2.3 Resource enumeration

In addition to secondary links, several 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 such as Digg, Reddit or Slashdot we gathered lists of active users on those sites by inspecting comment authors, and checked for profile pages of such users in the clients cache. Enumeration elements were particularly useful for tests where the same element might be visited (or input) 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.

[3]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

4.3 Submitting and 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.

5. RESULTS

The testing system based on this work was put into operation in early September 2009 and is currently available at [8]. Results analyzed here span the period of September 2009 to February 2010 and encompass data gathered from 271,576 users who executed of total of 703,895 tests (an average of 2.59 tests per user). The default `top5k` test, checking for 6,417 most popular Internet locations and 1,416,709 secondary URLs within those properties was executed by 243,068 users[4].

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 5 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 indeed susceptible to the attack.

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 Table 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.

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 reported significantly more detected web-

[4]The testing system was reviewed 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 5: Aggregate results for most popular tests

	# JS	# CSS	Found pri (JS)	Found pri (CSS)	pri/user (JS)	pri/user (CSS)	sec/user (JS)	sec/user (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
Adult	51311	1288	19.9%	49.6%	3.9	3.8	6.5	6.0

Figure 5: Cumulative distribution of `top5k` results for primary links.

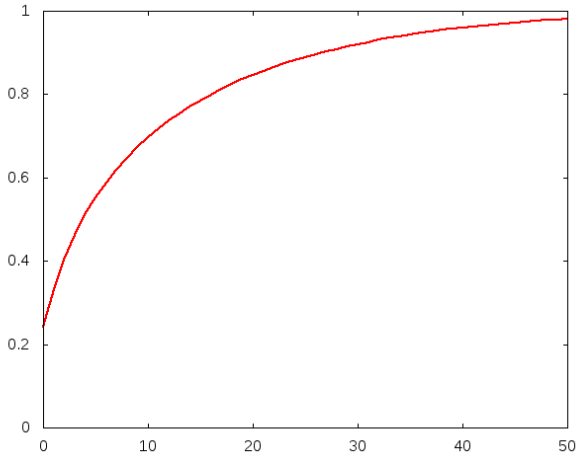

Figure 6: Cumulative distribution of Wikileaks results for secondary links.

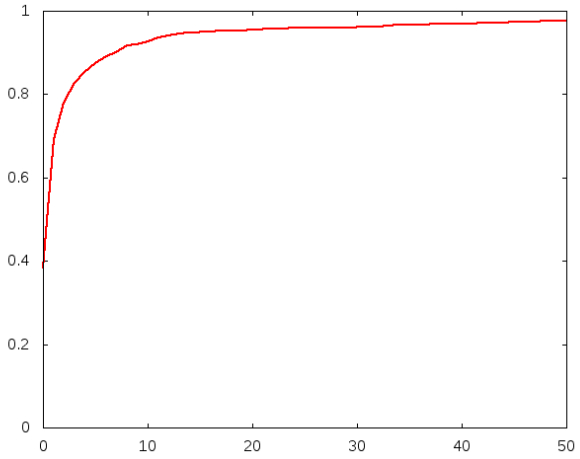

Table 6: News test results.

	Average secondary	Median secondary
All news	45.0	7
Digg	51.8	7
Reddit	163.3	26
Slashdot	15.2	3

sites, such as in the adult website test where we detected relevant links in histories of 49.6% non-JavaScript users compared to only 20% of JavaScript users. 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.

5.2 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 4.2.3.

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. 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.

The final news website test was the "All recent news" test which inspected the user's browsing history for 32 popular online news sites, along with links to stories gathered from their RSS feeds. For users who visited at least one such site, we found that the median of visited news story links was 7. For a significant number of users (2.9%) we detected over 100 news story links.

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 handpicked 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.

Additionally, we performed an analysis of detected secondary links on Wikileaks—a website with hundreds of potantially sensitive documents, in order to evaluate the feasibility of detecting Wikileaks users from a large body of our site visitors. We found that we could identify multiple users who had visited the Wikileaks site, some of whom had accessed a large number of documents, as presented in Figure 6.

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 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 is 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.

7. REFERENCES

[1] Bug 57351 - css on a:visited can load an image and/or reveal if visitor been to a site. `https://bugzilla.mozilla.org/show_bug.cgi?id=57531`.

[2] Cascading style sheets, level 1. `http://www.w3.org/TR/REC-CSS1/`.

[3] Cascading style sheets level 2 revision 1 (css 2.1) specification, selectors. `http://www.w3.org/TR/CSS2/selector.html#link-pseudo-classes`.

[4] Change the color of visited links. `http://www.useit.com/alertbox/20040503.html`.

[5] Quantcast. `http://www.quantcast.com/`.

[6] Stanford safehistory. `http://safehistory.com/`.

[7] Webcollage. `http://www.webcollage.com/`.

[8] What the internet knows about you. `http://www.wtikay.com/`.

[9] What they know. `http://whattheyknow.cs.wpi.edu/`.

[10] Yahoo! boss. `http://developer.yahoo.com/search/boss/`.

[11] Bug 147777 - :visited support allows queries into global history. `https://bugzilla.mozilla.org/show_bug.cgi?id=147777`, 2002.

[12] L. D. Baron. Preventing attacks on a user's history through css :visited selectors. `http://dbaron.org/mozilla/visited-privacy`, 2010.

[13] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 25–32, New York, NY, USA, 2000. ACM.

[14] E. K. C. K. Gilbert Wondracek, Thorsten Holz. A practical attack to de-anonymize social network users, ieee security and privacy. In *IEEE Security and Privacy*, Oakland, CA, USA, 2010.

[15] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 737–744, New York, NY, USA, 2006. ACM.

[16] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Commun. ACM*, 50(10):94–100, 2007.

[17] M. Jakobsson and S. Stamm. Web camouflage: Protecting your clients from browser-sniffing attacks. *IEEE Security and Privacy*, 5:16–24, 2007.

[18] F. König. The art of wwwar: Web browsers as universal platforms for attacks on privacy, network security and arbitrary targets. Technical report, 2008.

[19] H. W. Lie. Cascading style sheets. `http://people.opera.com/howcome/2006/phd/`, 2005.

[20] A. J. M. Jakobsson and J. Ratkiewicz. Privacy-preserving history mining for web browsers. In *Web 2.0 Security and Privacy*, 2008.

[21] T. N. J. Markus Jakobsson and S. Stamm. Phishing for clues. `http://browser-recon.info/`.

[22] W3C. Web content accessibility guidelines 1.0. `http://www.w3.org/TR/WAI-WEBCONTENT/#gl-color`, 1999.

[23] M. Zalewski. Browser security handbook, part 2. `http://code.google.com/p/browsersec/wiki/Part2`, 2009.