Abstract—For security and privacy on the Internet the webbrowser has become an important piece of software. Nowadays even mobile devices include a powerful browser, and in combination with cloud computing the webbrowser mimics more and more operating system functionality. For many use cases, legit server operators as well as attackers need to find out the exact version of a particular user's webbrowser. While the user agent string (which is currently used for that purpose) can be easily manipulated, certain aspects of the underlying software cannot. In our approach we will use the deviance of Javascript engine implementations to identify the browser accurately. This approach is efficient and has low computational overhead, and might be used to identify the version of the webbrowser regardless of the possibly manipulated user agent string.

I. INTRODUCTION

With drive-by-download attacks and privacy-violating tracking methods used by advertisement companies on the rise, attackers as well as defenders need to find out the exact version of the user's webbrowser. Legit websites use this information to mask implementation differences for the uniform display across webbrowsers. Currently, legit as well as malicious websites rely on the user agent string reported by the webbrowser in every request for optimization. However, this string can be changed easily and cannot be considered a security feature at all.

While webbrowsers evolve constantly, an arms race is currently going on between the top browser vendors. Firefox, Chrome, Internet Explorer, as well as Safari and Opera publish new releases in short intervals, implementing new features and accelerating the overall speed for downloading and displaying websites. Examples of such advanced features include a JIT compiler for the Javascript engine e.g., implemented in the Javascript engine of Google Chrome’s “V8”, or GPU support for Javascript in the current engine of Microsoft’s Internet Explorer “Chakra”. Mobile versions for smartphones and tablets are another area of constant improvements as these devices are becoming more and more popular among users. HTML5 and CSS3 will soon be fully supported by webbrowsers as these standards approach finalization, with more and more features that need to be implemented by the browser vendors.

Different use cases exist why a webserver needs to know the exact version of a client’s webbrowser, both benign and malicious:

- Appearance: for rendering a website correctly, different browsers have different methods. To provide a uniform appearance, webservers can provide different versions of websites for different browsers.
- User tracking: The webbrowser is a rather constant software component that does not change often. In combination with tracking HTTP or flash cookies this information of the exact browser version can be used to track users.
- Attack detection: If the webbrowser changes during a session this might be a hint for an attack against the user or the webserver as in the case of HTTP session hijacking.
- Attack surface: Current malware exploits vulnerabilities in webbrowsers to gain additional access on the victim’s system and install further software. To minimize the attack surface and increase success rate an attacker needs to know the exact version of the webbrowser.

Different methods exist for the webserver to get the version of the client’s webbrowser:

- User Agent String: The User Agent String is a field in the HTTP Request Header which allows the client software to set a string for the browser version. It represents the user agent, in particular the name, the operating system and the exact version.
• Navigator Object: The navigator object is part of every JavaScript implementation and includes the navigator.useragent, which contains the same information as the user agent string.

Our method uses the fact that implementations of the Javascript Engines differ from the standard. These differences form a rather unique fingerprint which can be calculated in very short time, and uniquely identify the browser.

II. BROWSER FINGERPRINTING

With the increasing use of the Javascript scripting language, most webbrowsers implement and support it. In fact, according to statistics by W3, in 2008 more than 95% of all browsers supported Javascript [1]. We will use Javascript for browser fingerprinting, respectively Javascript engine fingerprinting for browser detection.

As the webbrowser mimics more and more operating system behavior (e.g., the Google Chrome OS project), the proposed method mimics early operating system discovery as implemented with hugh success in nmap [1]. Nmap used the fact that the implementation of the TCP/IP stack was different for every operating system and exposed specific behavior. This is very similar to our approach. The reliability of the tests is in both cases very high, and as long as there are unique errors specific to a webbrowser and no untested browsers bias the results, there are no false-positives.

A. Preliminary Results

The Sputnik Javascript compliance test suite [2] will be used as starting point for collecting webbrowser fingerprints. It contains more than 5200 test cases to test to what extent the webbrowser’s Javascript engine covers the ECMAScript standard. While a full run of Sputnik takes between one and two minutes on modern hardware, the number of failed test cases is much smaller. In most cases the number of failed tests is well below a few hundreds, which means that an accurate version detection can be executed in less than 500ms. Chrome 13 under Ubuntu Linux has 136 failed test cases, while an early version of Firefox 3.0 on Windows XP has 270 failed test cases. It might be the case that a distinct test case is sufficient to uniquely identify a specific browsers, which could result in an execution time below 50ms. It has to be seen if this is possible, and how many tests are needed to identify the webbrowser while keeping the computational overhead and the execution time at a minimum. Depending on the changes in the Javascript engine, our approach must also cope with test cases that can identify the browser, but are not as specific as to uniquely identifying the browser. The number of failed test cases for various browsers can be seen in Table I. This list is not yet complete, as mobile browsers and Safari as well as Opera in different versions are missing, but it is a good start to show the cross-browser feasibility of our approach. The unique errors in the last column are relative, which means that within this preliminary dataset of seven distinct browser versions, one of the unique error is sufficient to uniquely identify the webbrowser in question.

<table>
<thead>
<tr>
<th>Webbrowser</th>
<th>Errors</th>
<th>Unique Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome 11</td>
<td>136</td>
<td>49</td>
</tr>
<tr>
<td>Microsoft Internet Explorer 7</td>
<td>475</td>
<td>4</td>
</tr>
<tr>
<td>Microsoft Internet Explorer 8</td>
<td>471</td>
<td>6</td>
</tr>
<tr>
<td>Microsoft Internet Explorer 9</td>
<td>72</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla Firefox 3.5</td>
<td>266</td>
<td>1</td>
</tr>
<tr>
<td>Mozilla Firefox 3.6</td>
<td>264</td>
<td>0</td>
</tr>
<tr>
<td>Mozilla Firefox 4.0</td>
<td>181</td>
<td>20</td>
</tr>
</tbody>
</table>

TABLE I: NUMBER OF FAILED TESTS FOR VARIOUS BROWSERS

B. Related Work

Relevant work in the area of browser identification and fingerprinting includes the Panopticlick project by the Electronic Frontier Foundation [3]. Their approach uses browser settings, installed plugins, fonts and screen size parameters to calculate the entropy of information transmitted to every webserver. A recent paper by Mowery et al. [2] proposes to use numerous JavaScript benchmarks and to generate a fingerprint for the timely execution pattern. Their approach is different in that a single run takes more than 190 seconds, and the time pattern for completing the individual benchmarks is used to identify the browser. Our approach is much faster (expected to be less than 1 seconds), and less error prone.

REFERENCES


[3] https://panopticlick.eff.org

http://www.w3schools.com/browsers/browsers_stats.asp
http://sputnik.googlelabs.com