BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Platform for measurement of JavaScript APIs usage on a web

Marek Schauer*

Abstract
The world wide web is a complex environment. Web pages can access many APIs ranging from text formatting to access to nearby Bluetooth devices. While many APIs are used for legitimate purposes, some are misused to track and identify their users without their knowledge. In this paper, we propose a methodology to measure the usage of JavaScript APIs on the public web. The methodology consists of an automated visit of several thousand websites and intercepting JavaScript calls performed by the pages. We also provide a design and architecture of a measurement platform that can be used for an automated visit of a list of websites. The proposed platform is based on OpenWPM. The browser is instrumented by OpenWPM and a customized Web API Manager extension is responsible for capturing JavaScript API calls.

Keywords: JavaScript — ECMAScript — Web API — API usage — Web measurement — Browser — OpenWPM

Supplementary Material: *N/A*

*xschau00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Web browsers offer a wide range of possibilities. A regular user can say that it is just a place for viewing web pages, but under the hood, web browsers provide an actual bridge between a viewed page and the host operating system. A web browser allows a web page to access information like values from sensors, information about battery status, installed fonts, and much more. Internet industry often takes advantage of such a wide range of information provided by web browsers and there is quite a high percentage of pages that collect a fingerprint from users and try to identify them to show them relevant content [1]. These practices are quite common and a large part of them is implemented using the JavaScript language.

JavaScript is a language that can be run in many environments, while the most well-known one is probably the web browser. Nowadays, there are solutions that provide JavaScript to the servers too, such as NodeJS. All these environments supply the basic set of ECMAScript language specified by the ECMA organization. In this article, we will focus on the web browser environment, which in addition to the base set of ECMAScript provides many additional APIs specified by other organizations (e.g. W3C or WHATWG). These standards are often called Web APIs and in this article, we will call these Web APIs as JavaScript APIs.

Many JavaScript APIs are used in a manner that has nothing in common with the original purpose of the API; this paper is concerned mainly with user tracking and fingerprinting. This behavior of websites usually includes using many distinct APIs, that provide some kind of specific information [1, 2]. For example Battery Status API implementation on Mozilla Firefox revealed very precise value allowing the trackers to identify the user for a period of time [3, 4]. Because the Battery Status API was used heavily for fingerprinting, it has been removed from Mozilla Firefox in 2017. Other examples of JavaScript APIs that are used often for fingerprinting are Canvas API [4], Audio API [2], Permissions API [1] or APIs for working with users device sensors [5].

In our work, we aim to measure the JavaScript APIs usage by popular websites. In this article, we

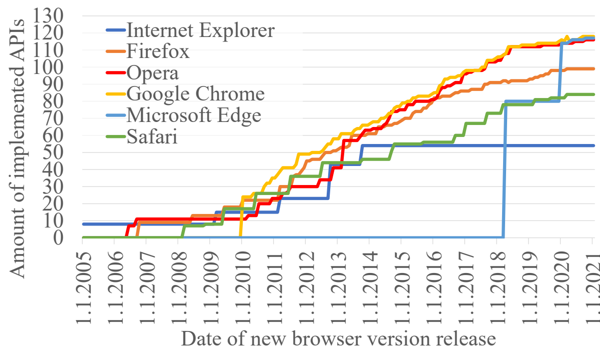

Figure 1. Progress of Web APIs amount implemented in distinct browsers in time.

present core technologies to accomplish these measurements. The stack of technologies is based on OpenWPM[1] enriched by a browser extension, that allows us to intercept JavaScript calls of different APIs. This browser extension is based on Proxy objects. Further details about core technologies are given in Section 3.

Our work is based on work of Peter Snyders et al. [6] in 2016. Since then many new APIs were specified and implemented in web browsers. Progress of APIs amount implemented in several most famous web browsers is illustrated in Figure 1. Figure is based on data from Can I use website[2].

In our work we will reproduce Snyders measurements with a focus on new JavaScript APIs. More details about results of Snyders research are given in section 3.1. In this paper, we propose the methodology of our own measurements in section 2. Further, in subsections 3.2, 3.3 and 3.4 we describe technologies that will be used for measurements. Finally, in section 4 we describe our contribution and explain how we customized and adjusted these formerly mentioned technologies to perform measurements.

2. Methodology proposal

This section describes the methodology that is planned to be used when the actual measurements will be performed. This methodology is based on the methodology used in previous research conducted by Peter Snyders [6]. Building a methodology on top of a methodology used in previous research can be profitable in two ways. First, the methodology is already validated by the authors of the mentioned study. Second, using a methodology that is very close to the original one can show us a difference in the usage of the JavaScript APIs between 2016 and 2021.

[1] https://github.com/mozilla/OpenWPM
[2] https://caniuse.com/

The main idea of the measurement is to visit several thousands of the most famous pages on the internet and intercept as many JavaScript calls as possible.

Visiting websites will be performed through the commonly used web browser, specifically Mozilla Firefox. This web browser will be enriched by an extension, that can intercept and log the JavaScript calls.

To maximize the amount of intercepted JavaScript calls we will visit not only the landing page but also the subset of subpages of each website. From the landing page, we will extract three links that point to a subpage of a given page. From each of these three subpages, we will get another three subpage links resulting in up to 13 pages of a given website being visited. This amount of pages should be high enough to catch the most of JavaScript calls.

To visit several thousands of websites we need to assemble a list of website addresses, that will be visited when performing our measurements. In this area, we will take advantage of previous work of other researchers and we will use the Tranco list. In the research conducted by Snyders, the Alexa Top Sites list provided by Amazon was used. Our decision of choosing the Tranco instead of the Alexa Top Sites is explained in subsection 3.4.

To give a web page a chance to perform all the JavaScript calls as it would be visited by a real user, we will wait and intercept JavaScript calls for 30 seconds on each page.

Results of our measurements should also provide information about the JavaScript APIs, that were probably used in a manner, that is not necessary for a page to be working and is very likely used in a way, that the user would not find useful. To achieve this, we will run our measurements on every page in two different modes. At first, we will visit the page using the common browser, which is not extended by anything special except the extension used for intercepting JavaScript calls and logging them. Then we will visit the same page using a browser complemented by an extension for blocking ads. The difference between logged data will show us which APIs were blocked by the ad blocker and which APIs were left out without blocking. As an extension for the ad-blocking, we picked a generic version of well known Ghostery extension for Mozilla Firefox.

3. Existing research and state of the art

This section provides key insights into the existing research in the field of JavaScript usage on the web. Further, in the following subsections, we describe the state-of-the-art tools and technologies that we plan

to use later when the measurements will be actually taken.

3.1 JavaScript usage on the web

The Snyders study suggests that some of the JavaScript APIs are extremely popular and they are used on more than 90% of measured pages (e.g. a well known `Document.createElement` method from DOM API). On the other hand, there are many APIs that are used by a minority of measured pages. That being said, almost 50% of JavaScript APIs implemented in the browser at the time were not used by any of the measured pages.

The study also suggests that there is no direct connection between the implementation date of a given JavaScript API in the browser (or by its specification date by some of the specifications vendors) and its popularity in using by websites. Concretely, there are some old JavaScript APIs, such as *XMLHttpRequest*, that are still very popular. However, there are also quite new JavaScript APIs, that are used very frequently (i.e., *Selectors API Level 1*).

The conducted study also measured the pages in two ways - with the ad blocker and without any ad blocking extension. Results of measurements showed that the blocking of different JavaScript APIs is not uniform and some APIs are blocked more often than others. Specifically, 10% of JavaScript APIs were blocked in 90% of cases resulting in a fact that 83% of APIs were used on less than 1% of websites when the page was visited with active blocking extension.

3.2 OpenWPM

OpenWPM is an instrumentation and automatization tool for visiting a large amount of websites [7]. This tool has been developed primarily for research in the field of web privacy and it is built on the top of Selenium, a tool for testing a front end of websites. While the Selenium itself is designed to test a single website usually by its owner or developer, OpenWPM helps to instrument Selenium while visiting a large amount of websites to perform some kind of measurements. In OpenWPM, Selenium is used to instrument the Mozilla Firefox browser. OpenWPM allows loading custom extensions into the web browser. In our work, we extend the Mozilla Firefox browser by a modified Web API Manager extension. The description of this extension is provided in following subsection 3.3 and the process of its modification is described in section 4.2.

Another benefit of using the OpenWPM tool is the fact that it allows parallelizing the websites visits by specifying the number of concurrent browsers that can

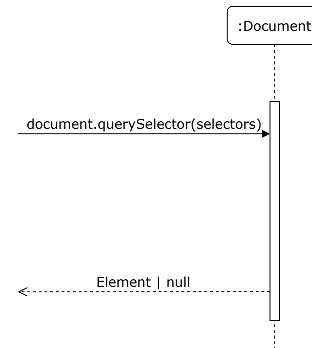

Figure 2. Sequence diagram of `querySelector` method called directly

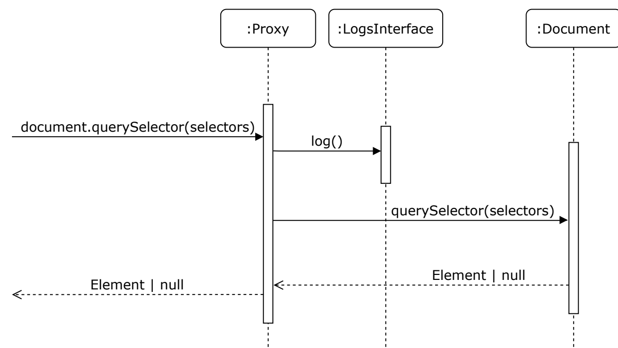

Figure 3. Sequence diagram of `querySelector` method calling through proxy object

be deployed. OpenWPM also offers the interface of a log aggregator to perform the logging. This log aggregator interface is exposed over the sockets and under the hood, it can save data coming from measurements into the database.

3.3 Web API Manager

Web API Manager is a browser extension, that aims to block explicitly defined JavaScript APIs. It has been developed by Snyders in 2016 and used in several studies conducted by Snyders et al. [6, 8]. The main principle of Web API Manager is based on Proxy objects. This metaprogramming technique allows intercepting calls performed on objects. While the main goal of the Web API Manager extension is to block the calls performed on objects that belong to particular JavaScript APIs, our goal is only to intercept these operations and delegate them to their original receivers.

The idea of intercepting method call using the Proxy object is depicted in figures 2 and 3. Figure 2 illustrates calling the `document.querySelector` method on the Document object, while the Figure 3 demonstrates the idea of calling the same method on the Proxy object.

The description of how this web browser extension is adapted for use in our measurement is provided in section 4.2.

3.4 Tranco

As stated in section 2, to visit several thousands of websites we need to provide a list of websites that will be visited when the measurements will be actually performed. There are several options when choosing between lists provided by different vendors, for example, Alexa Top Sites by Amazon, Majestic Million provided by Majestic, or Cisco Umbrella 1 Million by Cisco. Snyders methodology is based on visiting the websites provided by Alexa Top Sites list, however, the study suggests that these formerly mentioned lists are prone to various attacks and that they are quite unstable [9]. Additionally, unlike the two latter lists, the Alexa Top Sites list is no longer available for free. Another possibility when choosing the list of websites is a Tranco list, which aims to solve mentioned problems by combining presented lists and producing the new list. In order to use a list, that is free, immune to various types of attacks, and stable, we chose the Tranco.

4. Our contribution

To be able to run the measurements, we need to modify the presented tools and integrate them to work with each other. In this section, we describe these modifications.

In Figure 4 we provide a simplified illustration of the measurement platform. In the middle of the architecture illustrated in Figure 4, we can see the OpenWPM, which provides instrumentation of Selenium and Mozilla Firefox enriched by Web API Manager extension. Using OpenWPM and its instrumentation we can visit websites from the Tranco websites list and log the measured data to the database.

4.1 OpenWPM customization

To be able to intercept JavaScript calls performed by sites visited by OpenWPM, we need to enrich the OpenWPM infrastructure by logic that will allow us to do this kind of intercepting. In our case, this logic is placed in a web browser extension, concretely Web API Manager, which we need to integrate into the browser that is instrumented by OpenWPM to visit given sites.

There are two possible ways of integrating third-party extensions to the OpenWPM. The first way consists of placing the custom extension in a browser profile, that is loaded when the browser is launched by OpenWPM before visiting the site. This approach is used when loading the Ghostery extension in order to prevent the telemetry that is being interchanged after the fresh install of the Ghostery extension. Creating the first setup manually and packing the extension within the profile eliminates this undesired behavior. However, loading the whole profile is an additional operation that consumes additional time and memory resources. Another way of integrating the extension to the OpenWPM is to load the extension to the browser programmatically from Python. When loading the Web API Manager, we prefer the latter way of integrating the web extension into the browser because we can gain more control when the extension is being actually loaded and the operation of integrating the extension this way is faster.

4.2 Web API Manager customization

In its default configuration, the Web API Manager does not do anything. After the Web API Manager is loaded in a web browser, the extension is supposed to be further configured by a user. To intercept as many JavaScript calls as possible, we needed to modify this default configuration to intercept all JavaScript calls known to the extension.

The original purpose of the Web API Manager is to block explicitly defined JavaScript APIs. However, in our measurements, we just need to intercept the API calls, log this call and delegate the call to the original receiver. This behavior has been accomplished by modifying the system of Proxy objects definition and creation. Delegation to the original receivers has been implemented by using a JavaScript Reflect API.

Another challenge of modification the Web API Manager is how to log intercepted API calls into the OpenWPM database. The OpenWPM exposes the Log Aggregator interface through the socket interface. To connect to this logging interface via the socket, we use the infrastructure used by OpenWPMs default extension. The mechanism of logging the method calls is depicted in Figure 5. An instrumentation script responsible for the creation of Proxy objects is injected into the page content by the content script because content scripts do not have access to a global window object and therefore, cannot create Proxy objects that are eventually used by the web page. When the injected instrumentation script intercepts the method call, the event with a special name is triggered. The content script is instrumented to handle these events by sending the message into the background script. When the background script receives the message from the content script about a method being invoked, it propagates information about the method call via the socket interface to the OpenWPM. After that, the OpenWPM saves the information into the database.

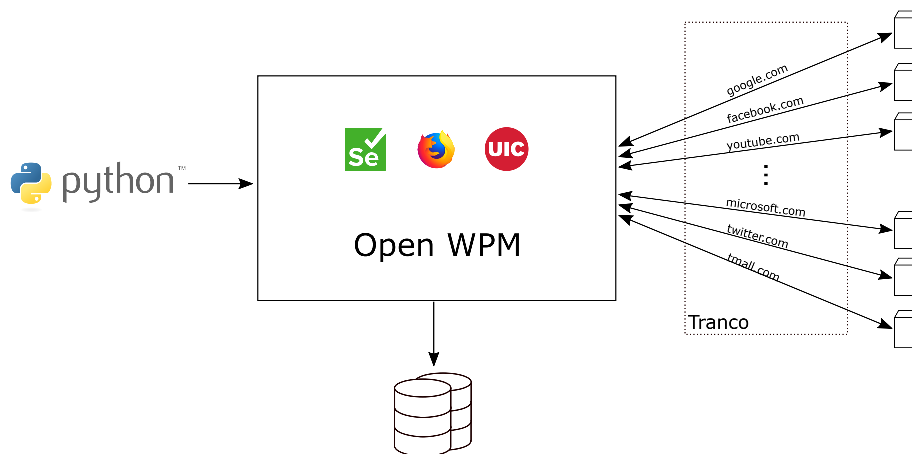

Figure 4. Measurement platform architecture. Architecture is based on OpenWPM enriched by the customized Web API Manager extension (illustrated as an UIC icon on the image) for intercepting JavaScript calls.

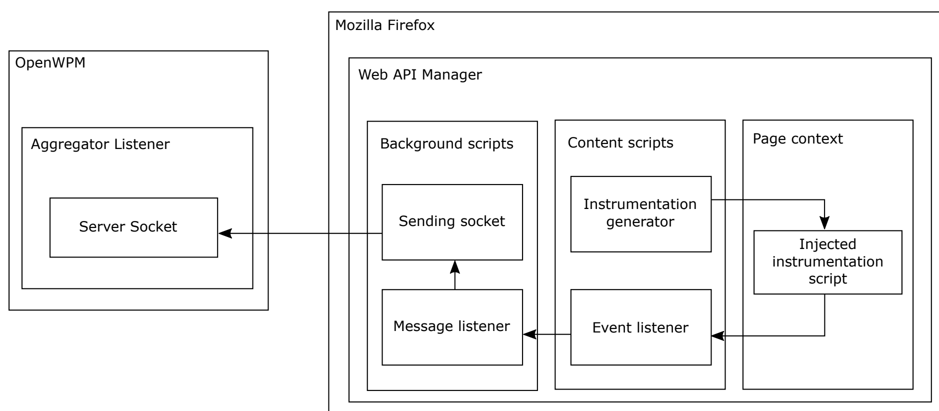

Figure 5. Integration of logging using Web API Manager and OpenWPM through socket interface.

4.3 Extracting information from Web IDL

To provide a Web API Manager the list of JavaScript APIs members we need a list of supported APIs. The actual state of the Mozilla Firefox is captured in special files, that are built in a compile time of the browser. These files are known as Web IDL and their main goal is to map the interfaces of JavaScript APIs available in the browser to the inner browser implementation. Specifically, it contains information about interfaces, their members, data types, and more.

The process of getting all the needed information from Web IDL files is fully automated. At first, we download all the Web IDL files from the Mozilla Firefox codebase. After that, we prepare these files for the parsing by a `webidl2.js` parser by W3C. Concretely, the `webidl2.js` parser is unable to parse the bodyless interfaces, so we extract these interfaces from Web IDL files and parse these modified Web IDL files consequently.

In a process of retrieving the list of all features implemented in the web browser, we need to distinguish between different types of interface members. For example, we want to omit the members, that have `ChromeOnly` attribute, which denotes a member, that is available only to the inner code of the web browser.

5. Current state and future work

The platform described in this article has already been implemented and in a time of writing, measurements are running. Measurements are running on the remote

machines provided by Digital Ocean. Using the Digital Oceans dashboard we created 14 so-called droplets with parameters presented in Table 1. The term droplet can be understood as a classic virtual private server.

Parameter name	Parameter value
CPU type	Shared
vCPUs	4
RAM	8 GB
SSD size	160 GB
Operating system	Ubuntu 18.04 (LTS) x64
Datacenter	Frankfurt

Table 1. Parameters of servers used to measure the JavaScript usage on the public web

When the measurement will be finished, the next stage of work will be to interpret the measured data. Measured data will be distributed in many databases over several servers and after saving them into the unified storage, we will abstract the data to contain information needed for the analysis. Concretely, once the measurement is finished, the SQLite database will contain one row for each JavaScript call. To make the manipulation with measured data more effective and convenient, we will abstract these data by creating a unified database, where only the number of calls of a given method on a given site will be saved. Further analysis of measured data will consist mainly of writing the scripts that will help us to see the patterns in measured data.

Thanks to the proposed methodology of crawling the websites in two ways, with and without the Ghostery extension, we will be able to identify which APIs are probably used in a manner that is not useful for the user of the website. Another interesting analysis might be performed by analyzing the order of the JavaScript calls as they were logged into the database in the same order as they had been performed.

6. Conclusion

The web is getting more and more complicated. Users browse their favorite websites from a wide scale of devices giving the website creators a wide range of possibilities through a wide range of JavaScript APIs implemented in their devices. Previous research shows that these APIs are not always used in a manner that is compliant with the original purpose of these APIs.

In this article, we proposed the methodology, that can be used to perform the measurements of actual JavaScript API usage on the web. We also provide an architecture of a measurement platform, that can be used to apply a given methodology. The measurement platform is based on OpenWPM extended by a customized Web API Manager. Further, we proposed the description of the changes that need to be done in these tools to integrate them into each other.

Acknowledgements

I would like to thank Mr. Ing. Libor Polčák, Ph.D. for his valuable advice, guidance, information, and support throughout the whole time of elaborating this work.

References

[1] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors, 2020.

[2] Steven Englehardt. *Automated discovery of privacy violations on the web*. PhD thesis, Princeton University, Chicago, Illinois, 2018.

[3] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. Battery status not included: Assessing privacy in web standards. *CEUR Workshop Proceedings*, 1873:17–24, January 2017. 3rd International Workshop on Privacy Engineering, IWPE 2017 ; Conference date: 25-05-2017.

[4] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[5] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. The web's sixth sense: A study of scripts accessing smartphone sensors. pages 1515–1532, 10 2018.

[6] Peter Snyder, Lara Ansari, Cynthia Taylor, and Chris Kanich. Browser feature usage on the modern web. pages 97–110, 11 2016.

[7] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of ACM CCS 2016*, 2016.

[8] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. pages 179–194, 10 2017.

[9] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, NDSS 2019, February 2019.