Console Clutter: A Cross-Browser Measurement of Console Messages

Thomas Helbrecht[1] and Jannis Rautenstrauch[2]

[1] Saarland University, Saarbrücken, Germany
research@thomashelbrecht.de
[2] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
jannis.rautenstrauch@cispa.de

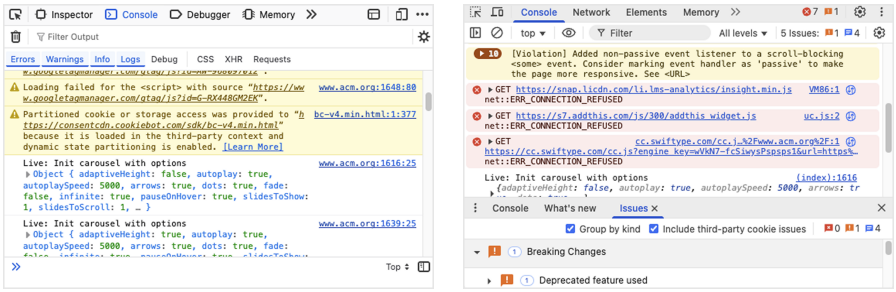
Abstract. Anybody who opened the browser developer tools on a random website was likely shocked by the wall of warnings and errors while the visited website appeared to function correctly. Although the developer tools and the contained *console tab* have existed in modern browsers for more than ten years and are regularly used by web developers and power users, there is limited information on the quantity and nature of messages developers encounter when opening the console, and whether these messages differ between browsers.
We close this research gap by performing the first systematic analysis of the browser console message ecosystem across the three main browser engines (Blink, Gecko, WebKit) on the landing pages of 51,984 websites. In total, we collected 2,088,416 messages, averaging 13.39 messages per website, with a maximum of 10,143 messages on one website in Firefox. While developer-caused messages (e.g., `console.log`) appear similarly across browsers, browser-caused messages (e.g., Cookie warnings or Content Security Policy errors) are strikingly different in both content and amount. With our in-depth analysis, we discover how console messages can reveal common misconfigurations and implementation issues. Lastly, we open-source our measurement framework to enable further research into the browser console message ecosystem.

Keywords: Console messages · Web measurement · Web debugging · Browser developer tools · Cross-browser analysis.

1 Introduction

In 2025, 6.04 billion people used the internet [15] to surf more than 1.3 billion websites [22]. To allow web developers to better debug websites, browsers introduced developer tools such as the browser console as far back as 2006 [12].

Anecdotal evidence shows that developers are annoyed by confusing messages and differences between browsers [24,10], however, there is no systematic evaluation of message differences across browsers. To close this research gap, we perform the first systematic analysis of the browser developer tools message ecosystem. We created a framework to comprehensively collect *console messages* available in the developer tools across all three major browser engines, collecting



(a) Console in Firefox (b) Console in Chrome

Fig. 1: Browser console excerpts when visiting `https://www.acm.org/`

both messages emitted by websites using the Console API as well as warnings and errors produced by the browser. We then perform a large-scale measurement study across 51,984 landing pages in all three browsers collecting a total of 2,088,416 messages out of which 1,557,329 are browser-caused and 531,087 belong to developer-caused Console API calls. During our in-depth analysis of the collected messages, we discovered significant browser differences with Firefox being more verbose in the total number of messages. We also reveal how console messages can be used as a proxy to discover implementation issues on websites.

To summarize, our paper makes the following contributions:

- We designed and implemented an open-source framework [13] to collect messages from the developer tools, which can differentiate between browser-caused and developer-caused messages (see Section 3).
- We performed the first large-scale measurement of the browser console ecosystem on 51,984 landing pages in Chromium, Firefox and Safari (see Section 4).
- We extracted insights from our in-depth analysis, discovering vast differences across the browsers in both type and amount of messages (see Section 5).

2 Background: The Browser Console

Early browsers included the *view source* functionality to allow users to see the HTML source code of a website as early as 1992. Later, many developers used `alert` calls to debug their applications [14]. However, with the success of the Web 2.0 more advanced debugging capabilities were required and the first browsers included what is now known as the *browser developer tools* in 2006 [12]. Although the feature set differs between browsers, all provide core functionalities such as a web inspector for viewing DOM hierarchy. Crucially, they include the *browser console*, which displays messages created using the Console API (e.g, `console.log`) and other messages about page problems such as JavaScript errors. Figure 1 shows the browser console when visiting the ACM homepage in Firefox and Chrome.

Modern browsers support different types of console messages (see Figure 1), visualized using different colors, icons, and additional information. The types of supported messages, their visual and internal representation is not standardized. Still, we can distinguish three types of console messages: *developer-caused messages*, *browser-caused messages*, and *page errors*, and describe the structure of these messages using the following three data points:

- **Content:** The content of the message that is displayed in the browser console. Modern browsers have pretty-printing capabilities and e.g., can render clickable URLs or display interactive JavaScript objects.
- **Log Level:** Each message has a log level, which determines the color the message is printed in, e.g., warnings are yellow. Additionally, modern browsers allow to filter messages by log level and prepend various icons for the different levels. The default log level is *log*, which has no color and no icon.
- **Source Location:** Each message has a responsible source location. This allows quick access to the source code emitting the message. Some general console messages have an empty source location.

Developer-Caused Messages A simple console message can be produced by calling `console.log("Hello World!")` from JavaScript using the Console API [26]. We label all messages from methods of this API *developer-caused* messages as they only occur if a developer invoked them. Aside from website JavaScript, browser extensions and service workers can produce those.

Browser-Caused Messages Apart from messages explicitly created using JavaScript, browsers also emit messages to inform developers about errors, signal failures in website functionality or announce feature deprecations. We call these messages *browser-caused* messages, as they originate from the browser implementation. Each browser vendor decides which messages to emit, and even messages on shared issues may appear different in terms of content and visualization.

Page Errors The third type of messages in the console are page errors. This category contains both JavaScript Errors [6] such as TypeError and DOMExceptions [27] such as NotFoundError. Page errors can be explicitly thrown by JavaScript code, e.g. using `throw new Error("err")`, or automatically thrown, e.g., when accessing non-existing variables. These messages are accompanied by a stack trace and can be caught using the `try...catch` construct.

3 Methodology: Measuring the Console Ecosystem

In the following section, we elaborate on our crawling pipeline, capable of extracting all information rendered in the developer tools.

3.1 Automated Message Collection

To automate the collection process of the browser console, we implemented our web crawler using the Playwright browser automation framework [18]. The framework's integration with each browser developer tools protocols' console event enabled access to all console messages including browser-caused messages. However, the captured console events do not distinguish between browser-caused and developer-caused messages. Thus, we retrofitted a mechanism for distinguishing browser-caused and developer-caused messages.

1. *Firefox:* The browser-caused console output follows a fixed message structure [20], which we used to identify browser-caused messages.
2. *Chromium & WebKit:* The console output does not provide clear information on the message originator, we patched the browsers to prefix messages originating from the Console API.

3.2 Template Extraction

Using our patched variants of Chromium and WebKit, along with the message structure for Firefox, allowed us to filter for browser-caused console messages. However, the collected messages contain dynamic data and lack information about the causing browser component. To recover the responsible *templates* (static strings with placeholders for data) and link them with their browser implementation, we developed a two-stage pipeline, first reducing the messages to template candidates by omitting dynamic data and then iteratively linking the candidates with the respective templates in the browser source code.

Dynamic Data Due to browser-caused messages containing dynamic data inlined into static templates, such as URLs, header values or numbers, we replaced these with placeholders. While we replaced all URLs in console output, we approached replacing header values by capturing all available values during a page visit and replace full header values if they occur in messages. Lastly, since we found that console messages also contained numeric strings signifying misconfigured values such as in WebGL details, we also replaced all numbers.

Recovering Templates To obtain the source code locations of the collected browser-caused messages, we used subsequences of each unique filtered message and queried the respective source code repository. For instance, by searching for *Loading failed for the <script> with source* in Firefox source code mirror on GitHub we were able to find the exact source code location emitting this message and extracting the full template. We iteratively performed this process until all browser-caused messages were matched to a template.

3.3 Template Categorization

In addition to collecting the source code location of each template responsible for browser-caused console output, we categorized the templates according to the

involved feature through manual inspection of the containing source code. Further, we grouped all categories into five high-level concepts: *Security-related (S)*, *Privacy-related (P)*, *Content-related (C)*, *Networking (N)* and *Other (O)*. For example, the *Loading failed for the <script> with source* template was categorized as *request-failures (N)*.

4 Results

In the following, we present the state of the browser console message ecosystem. We start with describing our measurement setup, proceed with an overview of the collected data and compare developer- and browser-caused output. We further expand on page errors, concluding with case studies.

4.1 Measurement Setup

In order to obtain a dataset of console output in the wild, we performed a crawl of sites taken from the Tranco list [16], generated on the 12th November 2024[3], using Playwright as the browser instrumentation framework. We partitioned the top 1 million list in buckets of size 100,000 and picked 10,000 sites uniformly at random for each bucket to select 100,000 sites. We visited each site once, collecting data from the landing page, on which we stayed for 30 seconds after loading. We used Chromium (v130.0.6723.31), Firefox (v131.0) and WebKit (v18.0).

4.2 High-Level Overview

We successfully collected data on 51,984 sites for all three web browsers. The low success rate originates from our use of an unfiltered Tranco list, containing many inaccessible sites such as `a-msedge.net`.

On the sites we visited, we captured a total of 2,088,416 console messages, with more than 78% of sites having at least one message. Table 1 shows the distribution of all messages, browser-caused, developer-caused and string-unique messages across sites and browsers. The total amount of messages in Firefox (1,396,566) is much higher than in Chromium (327,626) and WebKit (364,224), Firefox also recorded about 7,000 more sites with any message. This substantial difference in message counts is due to browser-caused messages, as all browsers produced a similar number of developer-caused messages.

4.3 Developer-Caused Console Messages

Our crawl yielded a total of 531,087 developer-caused console messages. WebKit was the most verbose with 189,528 messages. Notably, all three web browsers produced a similar amount of output of this overarching category, with the total spanning less than 18,000 messages apart on roughly the same amount of sites.

[3] Available at `https://tranco-list.eu/list/LJ274/1000000`

Table 1: Distribution of console messages per category

Type	Browser	Messages per Site			Total Messages	Number of Sites with Messages
		Mean	Median	Max		
All	Chromium	6.30	2	2,377	327,626	40,635
	Firefox	26.87	11	10,143	1,396,566	47,720
	WebKit	7.01	2	6,925	364,224	40,725
Browser-Caused	Chromium	3.83	2	2,375	155,729	31,441
	Firefox	25.71	12	10,142	1,226,904	47,012
	WebKit	4.29	2	2,395	174,696	34,961
Developer-Caused	Chromium	4.23	1	2,003	171,897	23,749
	Firefox	3.56	0	2,003	169,662	23,164
	WebKit	4.65	1	6,921	189,528	23,635
String-Unique	Chromium	3.66	1	848	99,419	40,635
	Firefox	13.20	8	1,367	296,238	47,720
	WebKit	3.94	2	518	90,349	40,725

Message Type Table 2 shows the distribution of types attached to developer-caused console messages, displaying the total amount captured and sites with at least one message. Overall, we collected 19 types of developer-caused console messages. The most common type by far was *log*, followed by *warning*, both occurring roughly the same across browsers. Comparing the other categories, substantial differences become apparent, in particular far less *error*-type messages were observed in Chromium. Likewise, the least amount of messages of type *clear* was collected in WebKit. Four types were never observed in WebKit and three never in Chromium. We verified that these are supported in all browsers, but are incorrectly reported as types *log* or *timeEnd* in Chromium and WebKit.

Message Site 52.26% of developer-caused messages originate from first-party scripts, and 47.74% from third-party scripts. The third-party messages belong to a total of 2,655 unique sites with the majority of messages belonging to a few key players such as Google and Facebook.

Message Content Out of the 531,087 collected developer caused messages, only 97,529 were unique (combination of type + message content). Messages related to the **JQMigrate** library appeared most often (20,095 messages logging the same version string). This was followed by developers logging objects from their JavaScript code (e.g. variables or DOM nodes). Some other common messages were error or log messages from libraries or frameworks such as odoo (OwlError), Google Maps API, or LiteSpeed. JavaScript types such as *undefined*, *RegExp*, or *function* were also commonly observed. We discovered that for many of the JavaScript types Firefox reports only the type name in the collected console message event whereas WebKit and Chromium report the stringified version. Note that the collected event and what is displayed in the console can differ, as the browser consoles use pretty printing. The pretty printing functionality differs across browsers, for instance, while Firefox displays a *RegExp* or *function* object interactively, Chromium and WebKit display a static string representation.

Table 2: Distribution of developer-caused console messages

Message Type	Total Messages			Sites with at least 1 Message		
	Chromium	Firefox	WebKit	Chromium	Firefox	WebKit
log	97,953	94,984	92,616	18,755	17,952	18,593
warning	24,790	24,725	28,093	6,990	7,154	7,287
error	11,970	16,569	21,992	2,764	2,568	2,810
debug	9,105	9,035	15,239	1,638	1,419	1,652
clear	13,349	11,036	3,176	119	58	54
info	6,129	6,566	8,258	2,536	2,561	2,693
endGroup	2,708	1,167	4,536	1,022	217	1,327
startGroupCollapsed	1,631	793	3,873	897	131	1,208
trace	1,075	1,065	3,285	291	283	555
table	1,894	921	2,585	110	97	372
dir	255	273	2,640	86	83	364
dirxml	215	234	2,466	62	59	334
startGroup	345	375	360	89	92	91
timeStamp	0	1,042	0	0	666	0
timeEnd	262	328	407	74	83	81
count	214	234	0	61	59	0
countReset	0	234	0	0	59	0
timeLog	0	79	0	0	2	0
assert	2	2	2	2	2	2

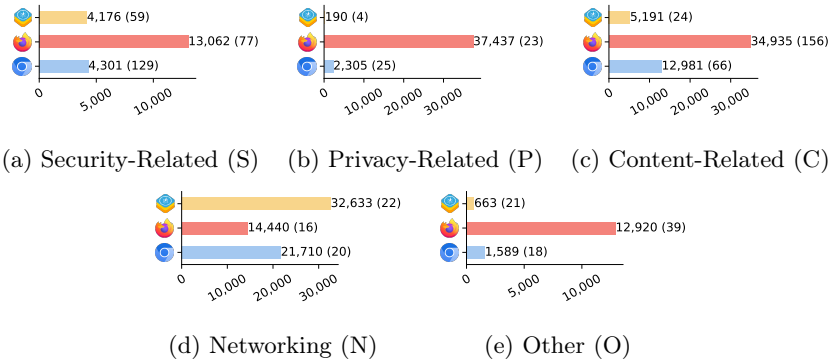
4.4 Browser-Caused Console Messages

Message Types Table 3 shows the distribution of browser-caused console messages types, displaying the mean, median, and maximum per site, and the total amount captured. It reveals several inequalities across browser vendors, with the first being that Firefox produced significantly more messages for the message type *warning* both in total and in the number of affected sites. Second, while Chromium and WebKit emitted a similar amount of console output typed as *error*, WebKit emitted substantially more *info* messages than Chromium. Moreover, although Firefox produced more *info* messages than WebKit in total, the number of sites with at least one *info* message is four times higher in WebKit.

Table 3: Distribution of browser-caused console messages

Type	Browser	Messages per Site			Total Messages	Number of Sites with Messages
		Mean	Median	Max		
warning	Chromium	1.27	0	1,028	66,179	13,125
	Firefox	19.45	9	8,471	1,010,973	46,635
	WebKit	0.75	0	764	39,230	9,481
error	Chromium	1.44	0	2,375	75,056	21,141
	Firefox	2.05	0	2,359	106,812	15,680
	WebKit	1.53	0	2,395	79,643	21,400
info	Chromium	0.17	0	70	8,898	2,703
	Firefox	2.10	0	357	109,119	4,460
	WebKit	1.07	0	855	55,823	18,267
verbose	Chromium	0.11	0	34	5,596	3,676
	Firefox	0.00	0	0	0	0
	WebKit	0.00	0	0	0	0

Only Chromium uses the *verbose* type. Messages of this type concerned mostly DOM-related hints regarding the creation of forms.



(a) Security-Related (S) (b) Privacy-Related (P) (c) Content-Related (C)

(d) Networking (N) (e) Other (O)

Fig. 2: Number of sites with browser-caused messages per browser across high-level concepts and constituting unique templates in parentheses.

Template Concepts To understand the involved browser components, we labelled each detected console message template with a category and a high-level concept. Figure 2 shows the number of sites having messages belonging to a concept for all browsers and the number of unique templates in parentheses.

Figure 2a shows that Firefox has messages related to security on most sites. Regarding privacy-related messages (see Figure 2b), Firefox produces output on most sites. Firefox also produces the most messages related to the content concept (see Figure 2c) and the catch-all other concept (see Figure 2e). For the networking concept (see Figure 2d), messages were produced on most sites by WebKit followed by Chromium.

Most Common Templates The 1,557,329 browser-caused messages belong to a total of 703 unique templates. In Table 4 we present the top-ten templates occurring on most sites for each browser (the full list of 703 templates is accessible online [13]). The most common template was the same for Chromium (**TC1**) and WebKit (**TW1**) and concerned the failure to load a resource. The most common template in Firefox (**TF1**) concerned the `SameSite` attribute of cookies. This template that occurred 707,776 times on 35,147 sites is by far the most emitted message overall and is mainly responsible for the huge verbosity of Firefox. However, other messages also occur more often in Firefox with the tenth-most popular template (**TF10**) occurring on 5,629 sites whereas **TC10** only occurs on 747 sites and **TW10** on 397 sites.

Three out of the top ten messages from Firefox (**TF1, TF2, TF5**) concerned cookie issues, while the other vendors top message templates did not concern any cookie related information. A commonality concerns the output of messages

Table 4: Top ten templates per browser

Id	Template	**Category** (Concept)	**Sites**	**Total**
	Chromium:			
TC1	Failed to load resource: Error	*request-failures (N)*	18,664	43,539
TC2	[GroupMarkerNotSet(crbug.com/242999)!: Number] Automatic fallback to software WebGL has been deprecated. Please use the –enable-unsafe-swiftshader flag to opt in to lower security guarantees for trusted content.	*webgl (C)*	5,995	9,977
TC3	The resource URL was preloaded using link preload but not used within a few seconds from the window's load event. Please make sure it has an appropriate 'as' value and it is preloaded intentionally.	*preload (N)*	4,493	17,720
TC4	[DOM] Input elements should have autocomplete attributes (suggested: " Name "): (More info: Static URL)(%o)?	*autofill (C)*	3,293	4,828
TC5	Slow network is detected. See Static URL for more details. Fallback font will be used while loading: URL	*styles (C)*	2,209	8,334
TC6	A preload for ' URL ' is found, but is not used because the request credentials mode does not match. Consider taking a look at crossorigin attribute.	*preload (N)*	908	1,971
TC7	Mixed Content: The page at ' URL ' was loaded over HTTPS, but requested an insecure element ' URL '. This request was automatically upgraded to HTTPS, For more information see Static URL	*mixed-content (S)*	901	10,830
TC8	Unrecognized feature: ' HeaderVal '.	*permissions-policy (P)*	808	828
TC9	Mixed Content: The page at ' URL ' was loaded over HTTPS, but requested an insecure Data ' URL '. This request has been blocked; the content must be served over HTTPS.	*mixed-content (S)*	770	1,752
TC10	Error with Permissions-Policy header: Unrecognized feature: ' HeaderVal '.	*permissions-policy (P)*	747	1,994
	Firefox:			
TF1	Cookie " Name " does not have a proper "SameSite" attribute value. Soon, cookies without the "SameSite" attribute or with an invalid value will be treated as "Lax". This means that the cookie will no longer be sent in third-party contexts. If your application depends on this cookie being available in such contexts, please add the "SameSite=None" attribute to it. To know more about the "SameSite" attribute, read Static URL	*cookies (P)*	35,147	707,776
TF2	The value of the attribute " Name " for the cookie " Name " has been overwritten.	*cookies (P)*	19,251	33,780
TF3	This page uses the non standard property "zoom". Consider using calc() in the relevant property values, or using "transform" along with "transform-origin: 0 0".	*styles (C)*	17,717	18,291
TF4	downloadable font: Glyph bbox was incorrect (glyph ids Data) (font-family: " Name " style: CSS weight: Number stretch: Number src index: Number) source: URL	*fonts (C)*	11,459	16,554
TF5	Cookie " Name " has been rejected for invalid domain.	*cookies (P)*	10,369	57,913
TF6	" URL " has been classified as a bounce tracker. If it does not receive user activation within the next Number seconds it will have its state purged.	*tracking (P)*	10,207	10,774
TF7	This page is in Quirks Mode. Page layout may be impacted. For Standards Mode use "<!DOCTYPE html>".	*html (C)*	8,321	12,420
TF8	The resource at " URL " preloaded with link preload was not used within a few seconds. Make sure all attributes of the preload tag are set correctly.	*preload (N)*	6,163	16,812
TF9	Layout was forced before the page was fully loaded. If stylesheets are not yet loaded this may cause a flash of unstyled content.	*styles (C)*	6,078	7,874
TF10	A resource is blocked by OpaqueResponseBlocking, please check browser console for details.	*request-failures (N)*	5,629	17,716
	WebKit:			
TW1	Failed to load resource: Error	*request-failures (N)*	19,168	44,959
TW2	Successfully preconnected to URL	*preconnect (N)*	18,266	55,822
TW3	The resource URL was preloaded using link preload but not used within a few seconds from the window's load event. Please make sure it wasn't preloaded for nothing.	*preload (N)*	5,497	16,782
TW4	window.styleMedia is a deprecated draft version of window.matchMedia API, and it will be removed in the future.	*styles (C)*	3,331	6,954
TW5	The page at URL requested insecure content from URL . This content was automatically upgraded and should be served over HTTPS.	*mixed-content (S)*	1,014	11,421
TW6	Origin URL is not allowed by Access-Control-Allow-Origin. Status code: Number	*cors (S)*	937	1,892
TW7	Refused to execute URL as script because HeaderVal is not a script MIME type.	*mime-type (C)*	865	1,201
TW8	[blocked] The page at URL requested insecure content from URL . This content was blocked and must be served over HTTPS.	*mixed-content (S)*	703	2,763
TW9	Viewport argument key " Name " not recognized and ignored.	*html (C)*	483	604
TW10	Failed to preconnect to URL . Error: Error	*request-failures (N)*	397	455

regarding preloading, where each vendor reported them among their most occurring templates per domains (**TC3, TF8, TW3**). Both Chromium and WebKit had many warnings about mixed content (**TC7, TC9, TW5, TW8**), while Firefox had no mixed-content related message in its top ten.

4.5 Page Errors

Table 5 shows the distribution of page errors based on their name across all three vendors, showing both their total count and the number of sites they were found on. We collected 92,518 page errors on 12,241 sites across all three browsers.

For both `TypeError` and `ReferenceError`, we detected errors on a similar amount of sites, although the totals differ significantly with Chromium having the most `TypeError`s. In WebKit, we collected 13,288 messages that are incorrectly reported as page errors by Playwright. Looking at the internal representation of console messages and page errors in WebKit, these messages are emitted with the type *JavaScript* and level *error*, which Playwright interprets as an page error even though they cannot be caught using try/catch [17]. We note that for developers, page errors and console messages of type *error* look identical on the WebKit console and cannot be easily distinguished.

If developers throw errors themselves, they are often using the generic *Error* type of JavaScript, however we also found a large number of developer-defined names, which we labelled as *Custom Error*. They have 63 distinct names, such as `AxiosError`. Another discrepancy is visible in errors named *SyntaxError*, where Firefox has significantly fewer.

Table 5: Count and unique sites per JavaScript Error

Error Name	**Chromium**		**Firefox**		**WebKit**	
	Total	Sites	Total	Sites	Total	Sites
TypeError	17,726	3,514	11,006	3,565	5,614	3,918
ReferenceError	8,760	1,818	9,640	2,071	3,700	2,078
ConsoleMessageLike	0	0	0	0	13,288	4,884
Error (Generic)	4,545	889	6,270	832	1,532	1,082
Error (Custom Name)	1,822	924	1,466	740	1,934	1,142
SyntaxError	1,538	1,062	578	272	1,502	1,198
DOMException	591	351	748	120	147	133
RangeError	37	5	30	1	34	5
EvalError	3	3	2	2	2	2
URIError	1	1	1	1	1	1
Total	35,023	7,190	29,741	6,559	27,754	11,429

4.6 Case Studies

Extending on the numbers, we investigated outlier sites and interesting patterns.

Outliers The site with the most browser-caused console messages, had a total of 10,143 messages in Firefox. We manually visited the site to ensure this was not a collection error and observed a similar amount of messages. 8,468 of those messages were about improper SameSite attributes of cookies and 1,670 rejections of cookies due to invalid domain. The crawler found the most developer-caused messages on a site in WebKit, where 6,919 out of the 6,925 of messages were produced by printing an OwlError, which can be attributed to the odoo library.

Local File Inclusions A potential privacy issue are console messages informing about the attempted inclusion of local files, revealing information about the developer's file system. An example console message leaking the file system structure of a developer is: `Not allowed to load local resource: file:// /C%7C/Users/sysw7/AppData/.../favicon.ico`. Similarly, we found attempts of probing for browser extensions: `Not allowed to load local resource: chrome://rumola/content/rumola48.png`. Tests for this particular resource were found on various government websites. The requested file belongs to the Rumola chrome extension, used for bypassing CAPTCHAs [1].

Anti-Debugging Techniques The Console API allows to clear the browser console using the `console.clear` function. Sites with a high number of these calls showed usage of anti-debugging techniques, e.g. through the inclusion of the JavaScript library *disable-devtool* [5]. The usage of such libraries aims to inhibit the use of the developer tools on websites. Notably, it fails to restrict data collection through our data collection pipeline.

5 Discussion

In this section we discuss the main insights gained by our results and how researchers and browser vendors can go forward improving the browser console for developers and using it for research.

5.1 Main Insights

Flood of Messages The maximum number of console messages on a site we observed was 10,143 and a total of 2,605 sites had more than 100 messages in at least one browser. With such high message quantities, the console ceases to be useful to developers and they might develop notification fatigue. Browser vendors seem to be aware of the issue and for example try to de-duplicate similar or identical messages. Unfortunately, Playwright does not expose details about whether console messages were merged with prior output. Furthermore, we found browser features aware of their verbosity, implementing mechanisms for suppressing excessive output. We found that Chromium suppresses further messages after four WebGL messages have been printed and WebKit stops after 256 such messages. Firefox implements a preference setting that allows the user to define the maximal number of WebGL warnings per context, which defaults

to 32. Most importantly, the Chrome team noticed that "the console was full of warnings and error messages" and introduced the Issues tab, an additional developer tool with structured messages, to declutter the console in 2019 [25]. However, we observed that this was only partially successful as there where still more than 332 sites with more than 100 messages in Chromium.

Browser Differences Our results show that there are significant differences in the amount and types of messages a developer encounters in different browsers. In particular, a developer using Firefox is exposed to more messages on average. Developers are only informed about certain issues in some browsers, and even if all browsers emit a message about an issue, the message itself is usually phrased quite differently. We also found that the implementations of browser console messages differs between browsers with Firefox colocating most of their templates in central files to allow for easy translation. In contrast, WebKit and Chromium opt for string concatenation directly in the code where an issue is detected, e.g., in the CSP parser if a CSP parsing errors is encountered. We also discovered typos and broken console messages in Firefox. For instance, we found that CSP messages were empty in certain conditions. The triaging process further revealed undefined messages for invalid *report-to* directive groups. Another inconsistency we observed was WebKit's direct passing of operating system library errors, such as for failed DNS resolution, into console messages.

Console Messages for Measurement Studies Our framework and collected console output reveals not only insights about the browser console message ecosystem, but also the state of the web, with console messages exposing implementation mistakes or hinting at general misunderstandings when considering them at scale. Web measurement studies can leverage console messages to pre-filter websites of interest, e.g., all with a certain message of type *error*. For example, studies on CORS [9], Mixed-Content [4], or XFO [3] implemented custom tools to collect data on these topics and often reimplemented browser functionality such as custom header parsers. If the browser-caused console messages emitted on these topics are detailed enough, e.g., for XFO all browsers report a message when an invalid header value is received, some research questions could be answered by using the console messages without the need of custom tooling.

5.2 Going Forward

Our results clearly indicate that there is no consistency and agreement across the browsers in how to implement console messages and for which events messages should be displayed. In addition, on many websites the amount of messages is really high, impeding effective usage of the console without deduplication. We propose further standardization of console messages with a centralized location of message templates to allow translation, user-friendlyness, and to ease studies on console messages. Browsers should provide complete console message data through their APIs to facilitate collection by tools like Playwright. Overall, we see this study as a first glance into the world of browser console messages. We

suggest to perform user studies to see what kind of messages (and which presentation) are helpful for developers similar to past work on end-user facing security warnings that improved significantly over the last few years [23].

6 Related work

Although the browser console was introduced in 2006 [12], there exists almost no research on it and we are not aware of any large-scale measurement on console output. Thus, in this section, we survey related work in the area of the browser console, web developer studies and browser security warnings.

Browser Console García et al. [8] developed a browser extension to collect browser console messages and evaluated their extension on 50 popular websites in Chrome, Firefox and Edge. Their evaluation shows that in addition to the *traceable* message categories they can collect, such as *console.log*, many other, often browser-specific, *non-traceable* messages such as security header warnings are emitted by browsers that cannot be collected by their browser extension.

We solve the issue of *non-traceable* messages by relying on Playwright's console event that has access to all emitted messages. We then perform the first large-scale measurement showing significant differences across browsers.

Web Developer Studies In several web developer user studies, the participants mentioned the use of the browser console or were annoyed or confused by inconsistent or difficult to understand messages highlighting the practical relevance of the browser console for web developers and the need for more research. In a survey of 109 JavaScript developers, 88.1% mentioned that they have seen console messages as a deprecation solution and 46% have used console messages to deprecate API [21]. More severely, more than half of the 12 participants of another study complained about inconsistencies in console messages, omitting helpful information depending on the involved browser vendor in a developer study about Content-Security-Policy misconfigurations [24].

Browser Security Warnings End-user facing browser security warnings have been studied much more than console messages [19,11,2,7]. In 2013, Akhawe and Felt [2] analyzed 25 million warning impressions collected from Mozilla Firefox and Google Chrome's in-browser telemetry data. Their study revealed differences in user behavior depending on the presented warning.

Our results on the inconsistencies of browser console messages across browsers highlight the need for further (user) studies on how browsers should design their developer facing console messages.

7 Conclusion

In this paper, we investigated the browser developer tools message ecosystem, comparing what information is displayed across Chromium, WebKit and Firefox.

For this, we developed a framework to comprehensively collect information from the developer tools, capturing console messages and page errors.

Analyzing a total of 2,088,416 console messages and 92,518 page errors across the landing pages of 51,984 sites, not only revealed a high amount of messages on many sites but also substantial browser differences. Moreover, our inspection of browser source code creating console output revealed inconsistent implementation approaches and a lack of coordination between browsers. These inconsistencies are worsened by differences in browser tooling, apparent in Chromium's Issues tab rendering information not necessarily present in the browser console.

As we consider this study a first glance into the world of browser console messages, we outlined research opportunities and open-source our framework [13] (MIT LICENSE, permanently available on Zenodo).

Acknowledgments. This work was conducted in the scope of a dissertation at the Saarbrücken Graduate School of Computer Science.

This version of the contribution has been accepted for publication, after peer review but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://dx.doi.org/10.1007/978-3-032-29372-5_6. Use of this Accepted Version is subject to the publisher's Accepted Manuscript terms of use https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Rumola - bypass CAPTCHA - Chrome Web Store (2022), https://chromewebstore.google.com/detail/rumola-bypass-captcha/bjjgbdlbgjeoankjijbmheneoekbghcg
2. Akhawe, D., Felt, A.P.: Alice in warningland: A Large-Scale field study of browser security warning effectiveness. In: USENIX Security Symposium (2013), https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/akhawe
3. Calzavara, S., Roth, S., Rabitti, A., Backes, M., Stock, B.: A tale of two headers: A formal analysis of inconsistent Click-Jacking protection on the web. In: USENIX Security Symposium (2020), https://www.usenix.org/conference/usenixsecurity20/presentation/calzavara
4. Chen, P., Nikiforakis, N., Huygens, C., Desmet, L.: A dangerous mix: Large-scale analysis of mixed-content websites. In: Information Security (2015). https://doi.org/10.1007/978-3-319-27659-5_25
5. Chen, T.: Disable-devtool (2025), https://github.com/theajack/disable-devtool
6. Ecma International: ECMAScript 2026 Language Specification (2026), https://tc39.es/ecma262/multipage/
7. Egelman, S., Cranor, L.F., Hong, J.: You've been warned: An empirical study of the effectiveness of web browser phishing warnings. In: ACM CHI Conference on Human Factors in Computing Systems (2008). https://doi.org/10.1145/1357054.1357219

8. García, B., Ricca, F., del Alamo, J.M., Leotta, M.: Enhancing web applications observability through instrumented automated browsers. In: Journal of Systems and Software (2023). https://doi.org/10.1016/j.jss.2023.111723
9. Golinelli, M., Arshad, E., Kashchuk, D., Crispo, B.: Mind the CORS. In: IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (2023). https://doi.org/10.1109/TPS-ISA58951.2023.00035
10. Gorski, P.L., Iacono, L.L., Wiefling, S., Möller, S.: Warn if secure or how to deal with security by default in software development?. In: IFIP International Symposium on Human Aspects of Information Security & Assurance (2018), https://pub.h-brs.de/frontdoor/index/index/year/2020/docId/4947
11. Harbach, M., Fahl, S., Yakovleva, P., Smith, M.: Sorry, I don't get it: An analysis of warning message texts. In: Financial Cryptography and Data Security. Springer (2013). https://doi.org/10.1007/978-3-642-41320-9_7
12. Hatcher, T.: 10 years of web inspector (2016), https://webkit.org/blog/5718/10-years-of-web-inspector/
13. Helbrecht, Thomas and Rautenstrauch, Jannis: Artifacts for Console Clutter: A Cross-Browser Measurement of Console Messages (2026), https://zenodo.org/records/18694496
14. Jay: A history of debugging on the web (2021), https://thehistoryoftheweb.com/checking-under-the-hood-of-code/
15. Kemp, S.: Digital 2026: Global overview report (2025), https://datareportal.com/reports/digital-2026-global-overview-report
16. Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M., Joosen, W.: Tranco: A research-oriented top sites ranking hardened against manipulation. In: Network and Distributed System Security Symposium (2019). https://doi.org/10.14722/ndss.2019.23386
17. Microsoft: wkPage.ts (2024), https://github.com/microsoft/playwright/blob/v1.48.1/packages/playwright-core/src/server/webkit/wkPage.ts#L538-L548
18. Microsoft: Fast and reliable end-to-end testing for modern web apps (2026), https://playwright.dev
19. Molyneaux, H., Kondratova, I., Stobert, E.: Understanding perceptions: User responses to browser warning messages. In: HCI for Cybersecurity, Privacy and Trust. Springer (2019). https://doi.org/10.1007/978-3-030-22351-9_11
20. Mozilla: nsScriptError.cpp (2024), https://github.com/mozilla/gecko-dev/blob/8fddaf56928e193b70c8e38973f6cd3dd5e3bb7d/dom/bindings/nsScriptError.cpp#L334
21. Nascimento, R., Figueiredo, E., Hora, A.: JavaScript API deprecation landscape: A survey and mining study. IEEE Software **39**(3) (2021). https://doi.org/10.1109/MS.2021.3103134
22. Netcraft: December web server survey (2025), https://www.netcraft.com/blog/december-2025-web-server-survey
23. Reeder, R.W., Felt, A.P., Consolvo, S., Malkin, N., Thompson, C., Egelman, S.: An experience sampling study of user reactions to browser warnings in the field. In: ACM CHI Conference on Human Factors in Computing Systems (2018). https://doi.org/10.1145/3173574.3174086
24. Roth, S., Gröber, L., Backes, M., Krombholz, K., Stock, B.: 12 angry developers-a qualitative study on developers' struggles with CSP. In: ACM SIGSAC Conference on Computer and Communications Security (2021). https://doi.org/10.1145/3460120.3484780

25. Scheffler, J., Schneider, S.: How we built the chrome DevTools issues tab (2020), https://developer.chrome.com/blog/issues-tab
26. WHATWG: Console: Living standard (2026), https://console.spec.whatwg.org
27. WHATWG: Web IDL Standard (2026), https://webidl.spec.whatwg.org/#idl-DOMException