Web Execution Bundles: Reproducible, Accurate, and Archivable Web Measurements

Florian Hantke[†], Peter Snyder[‡], Hamed Haddadi[⋆‡], Ben Stock[†]
† CISPA Helmholtz Center for Information Security ‡ Brave Software ⋆ Imperial College London
florian.hantke@cispa.de , pes@brave.com , h.haddadi@imperial.ac.uk , stock@cispa.de

Abstract

Recently, reproducibility has become a cornerstone in the security and privacy research community, including artifact evaluations [2, 35, 48] and even a new symposium topic [2]. However, Web measurements lack tools that can be reused across many measurement tasks without modification, while being robust to circumvention, and accurate across the wide range of behaviors in the Web. As a result, most measurement studies use custom tools and varied archival formats, each of unknown correctness and significant limitations, systematically affecting the research's accuracy and reproducibility.

To address these limitations, we present WebREC, a Web measurement tool that is, compared against the current state-of-the-art, *accurate* (i.e., correctly measures and attributes events not possible with existing tools), *general* (i.e., reusable without modification for a broad range of measurement tasks), and *comprehensive* (i.e., handling events from all relevant browser behaviors). We also present *.web*, an archival format for the accurate and reproducible measurement of a wide range of website behaviors. We empirically evaluate WebREC's accuracy by replicating well-known Web measurement studies and showing that WebREC's results more accurately match our baseline. We then assess if WebREC and *.web* succeed as general-purpose tools, which could be used to accomplish many Web measurement tasks without modification. We find that this is so: 70% of papers discussed in a 2024 web crawling SoK paper could be conducted using WebREC as is, and a larger number (48%) could be leveraged against *.web* archives without requiring any new crawling.

1 Introduction

Security and privacy Web measurement is an enormous field that lacks accurate, general, and standardized measurement tools and archival formats needed for reproducible measurements and general-purpose datasets. We argue that existing tools in the field operate at the wrong level of abstraction and granularity. In the best of cases, this mismatch invites error; in many cases, it makes certain categories of error unavoidable.

As a demonstrative example of how the most common current Web measurement tools and Web archive formats invite errors, consider the following example. Say a researcher wanted to measure how the Web behaved in 2010 (the first year the HTTPArchive project [72] provides archives). The common way such measurements are done is to use a browser automation framework like Puppeteer [17] to automate a browser and load a website from a structured HTTP archive such as a HAR file. Current implementations of Puppeteer use the current version of Chromium by default and are incompatible with very old browser versions, meaning that websites archived in 2010 will be executed in a 2024 (or later) version browser. One large change that occurred in the Web platform between 2010 and 2024 is that browsers changed how they handle certain "mixed content" resources, or insecure sub-resources included on a secure website. Before 2015, Chromium *would* load insecure JavaScript included on a secure page, but (in alignment with other browsers) after 2015, Chromium *would not* load it on a secure page. The net effect of this change is that measuring how the Web behaved in 2010 by using modern Chromium versions will cause several subtle but categorical measurement errors. It will under-count JavaScript sub-resource requests while over-counting the number of failed HTTP requests, among other errors. This is just one example.

Current measurement tools can potentially cause many other errors of this kind, as well as other entirely different classes of errors. Using current archival formats for historical Web measurements will cause comparable problems. More broadly, the lack of accurate, general-purpose Web measurement tools and archive formats weakens the field of Web security and privacy measurement in at least the following ways: *i.* by encouraging, or sometimes requiring, measurement errors; *ii.* by making it difficult to compare results across measurement studies (since differing results could be introduced by changes in measurement tools, measurement vantage point, etc.); *iii.* by preventing accurate and reproducible historical measurements; and *iv.* by consuming an enormous amount of researcher time on developing similar-but-bespoke measure-

ment tools for each task.

Research areas benefit greatly from having common, reusable datasets. For example, such datasets encourage incremental improvements, and allow the results to be easily, objectively, and accurately compared across papers. A primary reason why the field of security and privacy Web measurement lacks such datasets is that the field lacks a general-use, accurate format for archiving Web measurements, which would allow a common dataset to be reused across measurement tasks. By comparison, such datasets exist in other areas of computer science research, such as machine learning and system performance.

In this work, we aim to improve the state of security and privacy measurement in two ways. First, by presenting WebREC, a system for accurately measuring how browsers execute websites, in a manner robust to page circumvention, and covering a broad enough range of browser behaviors (e.g., rendering, network, script execution, etc.) that WebREC can be used without modification for extensive measurements. And second, with *.web*, a novel archival format for recording how a browser fetched, rendered, and executed a website.

WebREC and *.web* bring the following benefits for precise and reproducible Web measurement. First, they provide **comprehensiveness** where current tools do not. WebREC is deeply integrated with Chromium and is built with industry-assisted knowledge of browser architecture to ensure both comprehensive and correct results (see Section 2.2.4; this differs significantly from current Web measurement tools).

Second, WebREC and *.web* enable **accurate** archiving. WebREC records both the resources that were captured during page execution, along with a fine-grained execution timeline. *.web* archives enable researchers to ask a wide range of questions about how websites behaved at the time-of-archiving in a way fundamentally not possible with most current tools.

Third, WebREC and *.web* are **general** tools, and allow researchers to measure a wide range of browser behaviors without requiring modifications or customization. By aiming to be general and reusable, WebREC benefits from iterative improvements and corrections, in a way uncommon to most measurement tools (which, historically, tend to be single-use and discarded after each project).

More broadly, in this work, we make the following contributions to the goal of accurate and reproducible security and privacy Web measurements:

1. a **comprehensive overview of the Web measurement and archiving tools** commonly used in research, along with the limitations why each is insufficient for the goal of generating accurate, reusable, achievable Web data;
2. the **design of WebREC and *.web***, a reusable tool and format for measuring and archiving how browsers execute websites accurately and robustly to allow reproducible measurements beyond what current tools and archival formats provide;
3. multiple **empirical evaluations of WebREC's and *.web*'s accuracy**, compared against existing commonly used tools, finding that WebREC more accurately matches baseline counts of events;
4. an **empirical measurement of WebREC's generality**, by surveying recent security and privacy Web measurement papers and evaluating which could have been performed with WebREC directly, or *.web* archives, avoiding the need to create new, custom measurement tools;
5. the **open source implementation** of both, as modifications to Chromium (see Section 7) and related tools for crawling with WebREC, generating *.web* archives, and querying them regarding security and privacy-relevant phenomena (e.g., to query what Web APIs were called in which frames, among many other possibilities);

Finally, we plan a **publicly available archive** in the *.web* format, enabling researchers to conduct a wide range of accurate, replicable measurements of historical Web behavior. We pledge to regularly update this archive to create a common resource for Web research comparable to the HTTPArchive.

2 Tools & Techniques for Web Measurements

In this section, we provide a brief overview of existing tools, techniques, and formats commonly used in Web security and privacy measurements, based on a literature review of top-tier papers, such as those listed in crawling SoK and survey papers [1, 63]. Our goal is to be demonstrative of the common approaches used in research and discuss why these are insufficient for the replication, accuracy, and iterative needs of scientific research. We then follow by describing common formats used for archiving websites, so that measurements can be done retrospectively, after websites have been changed or are no longer available on the Web. We highlight each approach's strengths, as well as its limitations that make it insufficient as an accurate, general-purpose archiving format.

2.1 The Web and Browsers

Before discussing tools and formats for Web measurements, we briefly outline how the Web and browsers function.

Users typically control a client, such as a browser or curl, to request a resource from a Web server via HTTP requests. The server processes these and returns an HTTP response containing the requested resource and further metadata [38].

When browsers retrieve an HTML document, they parse it based on a well-defined specification [76]. This process involves tokenizing the input stream and constructing the Document Object Model (DOM) from it – a tree-like representation of the page's content. CSS is parsed into the CSS Object Model, combined with the DOM to build the Render Tree. From this tree, the browser calculates element positions and dimensions before painting them on the screen.

As the DOM is processed, additional resources may be loaded and JavaScript (JS) executions can modify the DOM.

These resources and dynamic changes are not part of the initial HTTP response and require rendering and further requests. Once all resources are loaded, the JavaScript is executed, and the Render Tree is finalized and painted, the user sees the Web page as the final product [37]. To measure and analyze these behaviors for research, tools and archives are needed that accurately interpreting these standardized processes.

2.2 Existing Web Measurement Tools

We first describe four general categories of tools commonly used in Web measurement. Based on our literature research, we find that all Web measurement tools used in top security and privacy papers fall into one of these categories.

2.2.1 Recording HTTP Requests

The simplest, and earliest, approach used in Web measurement research is to record the HTTP communication for each page being measured. In this approach, researchers select a tool for issuing an HTTP request to a URL (e.g., curl [11], wget [16], among many others), use the tool to request the HTML document for the webpage being measured, and record the outgoing HTTP request and the server's HTTP response. Numerous examples can be found in literature [3, 7, 25, 50].

This approach can be extended to use the same tools to record the HTTP communication for sub-resources (e.g., images, or script files) referenced in the HTML of the webpage, either by searching for URL patterns or parsing the HTML.

The defining feature of this approach is that tools are being used to understand or approximate how users experience the Web, but without actually incorporating the main tool people use to interact with the Web; Web browsers.

Limitations: Measuring the Web based on raw HTTP request data has important benefits; hundreds of instances of `curl` can be run in the amount of resources needed to load a single webpage in Chrome. However, this approach also comes with serious limitations; Evaluating HTTP and HTML without using a Web browser will give, an extremely *lossy* approximation of peoples' Web experience, making it unsuited for answering many Web related research questions.

As example, Web browsers use complicated rules for deciding which sub-resources to fetch, affected by many factors like a page's *Content Security Policy* [79], `preload` [75] instructions given on previous pages, among others. Concretely, consider a dynamically added image element; the image is typically fetched when the element's `src` attribute is set, regardless of whether it has been appended to the DOM. These edge cases require an extreme amount of domain-specific expertise, leading to accidental but *preventable* errors.

In short, tools in this category focus on HTTP communications but do not consider how Web browsers parse and execute those websites. Therefore, they are only suited to answer questions about server responses, but not accurate enough to measure how the Web is experienced by most people.

2.2.2 Browser Instrumentation

A second approach is to leverage the browser itself, either through automation APIs (i.e., WebDriver [77] and DevTools [10]) and popular libraries for interacting with these APIs (e.g., Puppeteer [17]), or through browser extensions. These approaches are used in a variety of Web research [12, 46, 49, 54] and account for the limitations of the previously discussed approach by measuring websites as executed by browsers.

All of these browser instrumentation techniques are able to measure how a website is rendered, cookies and other values are stored, and execute and inspect JavaScript in the page by injecting proxies into global JavaScript structures. Some tools also measure network requests made by the page.

Limitations: While these approaches are extremely common in Web measurement research, they have important limitations, making them unsuited for measuring certain phenomena researchers investigate. Broadly speaking, browser instrumentation tools lack APIs to directly measure many page behaviors, requiring researchers to try and write code that replicates browser behavior. In the best of cases, this invites *preventable* implementation errors; in many worst cases, measurement errors are unavoidable or *unpreventable*.

For example, browser-instrumentation-based measurement tools are not able to observe JavaScript execution directly, requiring researchers to rely on indirect best-effort techniques (e.g., modifying the execution environment). These techniques lead to incomplete or incorrect results for a range of reasons. Measurements that rely on modifying a script's execution environment fail when page scripts regain access to unmodified JavaScript prototypes (e.g., by storing a reference to the original prototype early during page execution), something extremely difficult to prevent and a pattern utilized by real-world advertising libraries or anti-debugging techniques [43]. Furthermore, measurements relying on interposing of non-configurable global JavaScript elements cannot capture the interaction with these (read-only) Web API properties, like `window.location`, making errors *unpreventable* and leading to a lack of *comprehensiveness* with current tools.

These are just some examples of phenomena that browser-instrumentation-based measurement tools have difficulties in *accurately* measuring. While suitable for functional tests (the task these tools were initially designed for [55]), these tools have significant limitations preventing them from being the *general* purpose solutions in Web measurement research.

2.2.3 Browser Orchestration

A third category of Web measurement tools builds on browser-instrumentation tools, relying on the same browser capabili-

ties and intervention points but creating reusable and improvable instrumentation code for consistent measurement tasks. This allows a common body of code to improve over time by benefiting from the collective knowledge of multiple researchers. In short, this third category of Web measurement tools improves measurement accuracy by pushing the level of abstract up another layer, allowing researchers to conduct measurements more easily and accurately without needing to personally understand the peculiarities of the Web platform.

The most popular example of this category of Web measurement tool is *OpenWPM* [15], used by studies all across the community [3, 8, 27, 82]. In addition to the main benefit of providing an open and reusable code base that researchers can use and improve, *OpenWPM* also has additional benefits. First, *OpenWPM* provides standardized inputs and outputs for conducting measurements, further aiding replication. And second, it also includes scheduling functionality, making it easier to measure websites in parallel across different machines.

Limitations: *OpenWPM* relies on the same underlying systems and capabilities as the previously discussed browser-instrumentation-based tools and so shares many of the same limitations. While *OpenWPM's* reusable, iteratively-improved code base reduces the chances that researchers will make *preventable errors*, *OpenWPM* and similar systems fundamentally cannot address the *unpreventable errors* caused by the limitations of the underlying capabilities OpenWPM relies on (see limitations in Section 2.2.2), e.g., not *accurate* enough to *comprehensively* measure all Web API properties.

2.2.4 Browser Modification

A fourth approach to Web measurement is to try and avoid the limitations of the APIs available to *browser instrumentation* approaches by directly adding instrumentation to the browser's code base, for example the DOM construction. Since all popular browsers are (in part or in full) open source [42, 71, 73], researchers can accurately measure any browser functionality by modifying their code.

This browser-modification approach has been used for a wide range of measurement tasks, generally to measure lower-level browser behaviors that are not directly visible to page-executed JavaScript or the APIs available to the Puppeteer libraries. Researchers have published research depending on browser modifications to log script compilation and fine-grained JavaScript execution [28], detect XSS attacks [68], the behavior of injected style sheets [33], and to perform taint analysis in JavaScript code [31], among many other examples.

Limitations: Measurement tools that modify how browsers are implemented provide researchers with powerful measurement capabilities. However, this flexibility imposes a wide range of limitations on these tools. Primarily, the correct implementation of such tools requires an extremely high amount of domain knowledge in the implementation and esoteric standardized behaviors of modern browsers, making *preventable* errors easy when instrumenting target behaviors. And while some of these errors may be obvious to researchers, e.g., build errors, other errors can be extremely subtle, and require domain expertise to catch, e.g., miss-attribution of edge cases.

As a demonstrative example of the difficulties of correctly implementing browser-modification-based measurement tools, consider the task of modifying Chromium to log access to browser APIs (the goal of several projects, such as [26, 28]). Measuring which script is calling which API *completely and correctly* requires understanding i. the low-level implementation details, (e.g., the differences between a V8 `Isolate`, a V8 `Context`, a V8 `ScriptState`, and a blink `ExecutionContext`) ii. subtleties in the browser application model (e.g., the difference between the calling and receiving execution contexts), and iii. a comprehensive knowledge of all code interacting with these details. This task is not uniquely difficult, and comparable difficulties exist for instrumenting the parsing behaviors, JavaScript ordering, among others.

We emphasize that these *are not* criticisms of existing research tools, which often aim for a general understanding of one phenomenon rather than complete correctness. Instead, we note these difficulties to show how current tools are not well suited for the tasks of a *generalized* Web measurement tools, i.e., they are not reusable for everyone for a wide range of measurement tasks without modifications.

2.3 Existing Web Archiving Formats

We next provide a brief overview of common archiving formats used in Web measurement.

2.3.1 URLs and Raw Resources

The simplest archive form generated by Web crawls is a list of visited pages' URLs, allowing revisits and re-measurements in the future (e.g., the Tranco list [47]). Lists like this are used to analyze phenomena like typosquatting [69] or similar.

Other archival datasets might include the actual resources described by the URL at measurement time, such as an image dataset containing both the URLs and the image's content. Including both makes the dataset more robust, since, even if the hosting server no longer returns the resource from the URL, the resource of interest is still available in the dataset.

Limitations: Such datasets are popular because they are easy to generate and to use. However, this approach has profound limitations, making it poorly suited as a *general* approach to Web archiving. Most fundamentally, such datasets lack many other resources and information present when the website was loaded and rendered (e.g., HTTP headers), limiting the ability to ask new questions about old data.

2.3.2 HTTP Archives with Metadata

The other common form of Web archiving is to record the HTTP headers and bodies of all requests and responses during

the page's execution, along with metadata like timestamps. Frequently used formats are the HTTP Archive format (HAR) and the Web Archive format (WARC), which record largely the same kinds of information, though with minor differences, such as WARC de-duplicates requests while HAR does not.

Sites archived in these formats record the communication while executing a target Web page. Researchers use them to replicate past measurements [20], or perform new ones on historical Web versions [34,46,51,67]. They replay responses by loading the initial request and the loaded sub-requests from the archive instead of from the network. Effectively, researchers attempt to re-run the Web site from the archive to approximate how the site operated at measurement time.

Limitations: These archives are very useful for measurements that study the resources loaded by a page, as sub-resources are archived exactly. While extremely common in Web privacy and security research, these approaches, however, are significantly *inaccurate* when trying to replicate how websites behaved during execution.

There are multiple reasons why these archive formats do not accurately enable future researchers to replicate how archived websites behaved. For example, the web platform involves many sources of non-determinism (e.g., the current time, random values, available system resources, etc.) which can cause websites to behave differently on re-execution – an *unpreventable* issue when this behavior was not captured.

Similarly, changes in browsers and archiving tool behavior over time further introduces errors and inaccuracies. Browsers' behavior at *archiving* time may differ significantly from how browsers behave at *replay* time, introducing entire categories of potential inaccuracies. For example, consider the aforementioned change of the *mixed content* policy. Modern browsers block insecure JavaScript files while browsers ten years ago had no such restrictions [78]. This change means replaying such archives in modern Web browsers will cause categorical inaccuracies, making it seem like users historically encountered far more blocked requests than they actually did.

This change is just one example of how such archives can lead to systematic errors in historical Web measurement. Differences in the *archiving* and the *replaying* environments in any of the following will cause measurement errors: differences in i. browser (e.g., Firefox v.s. Chrome), ii. browser version, iii. operating system, iv. updates in available Web APIs, v. *headless* or *headed* browser modes, and vi. browser language and locale preferences, among many others. These errors might be *preventable* when the exact infrastructure is replicatable, in many cases it is not [13].

2.4 Summary: Limitations and Requirements

Finally, we note the overall challenges facing researchers attempting to conduct accurate, replicable, shareable Web research, given the limitations in existing Web measurement and archiving tools, leading to requirements for a new tool.

Prohibitive Required Domain Expertise: The vast majority of non-trivial measurements of Web behaviors require a large amount of domain expertise. *Accurate* measurements demand understanding and often re-implementing various browser behavior, challenging even for the industry teams implementing browsers, let alone academic Web researchers.

In the best of cases, the expertise needed for *accurate* measurement means researchers must familiarize themselves with esoteric browser concepts to *accurately* approximate how users experience the Web. In the worst of cases, the high level of required expertise results in incorrect results.

Redundancy and Correctness: The current Web measurement field lacks tools that are both *general* for use across a wide range of measurements, and implemented in a manner that yields *accurate* results, i.e., avoid aforementioned *comprehensiveness* limitations (Section 2.2.2). Instead, projects typically implement a new *instrumented browser*, typically without correctness tests beyond checks by the researchers themselves. The systematic result of such redundancy is a *worst of all worlds* situation: a lot of time spent (i.e., research teams re-implementing similar tools) creating tools of unknown correctness (i.e., often no correctness verification).

Limited Dataset Sharing: Despite the enormous amount of Web measurement work conducted and published, our field lacks *general* datasets for *comprehensive* measurement tasks. The closest our field has are URL lists or collections of archive files, both with discussed limitations. The lack of a general dataset of website behavior hinders iterative improvements, as seen in other fields with common datasets like AI or ML.

Tool Requirements: As a result, Web measurements result in independent executions of (often) different websites in independently implemented measurement tools, making it difficult to confidently compare results across papers.

To address this, a new tool is needed, especially to handle the *unpreventable* errors from Web API restrictions and replicability problems with dynamic behavior. We developed the following requirements for such a tool. First, it must be *comprehensive* and support a wide range of measurements out of the box, including JavaScript executions, without requiring specialized domain expertise. Second, it must be *accurate*, also when handling JavaScript observations. And third, the tool must be *general*, producing reusable archives that go beyond only storing a site's resources but also provide all the executions to enable accurate replication of experiments.

3 WebREC and *.web* Archives

In this section, we introduce **Web Execution Bundle RECorder** (WebREC), a tool for accurately measuring a wide range of browser behaviors during the execution of a webpage, and *.web*, a three-part archive bundle format that WebREC uses for recording its observations. The following subsections detail the designs of WebREC and *.web* and how they overcome the limitations of existing tools (Section 2).

3.1 Overview and Goals

WebREC is designed to be *accurate*, *general*, and *comprehensive*, i.e., it can capture a wide range of browser behaviors and events, making it suitable for a large fraction of security and privacy Web measurements without the need for modification; we expect that data necessary for diverse studies is a subset of what WebREC currently measures.

WebREC records its measurements as a single *.web* archive, which is a bundle of three child files, described in more detail in the following subsections. The format of these files is consistent and well structured so that common tooling can be used to query the page behaviors and events recorded in each *.web* archive, the same way that SQL databases describing widely differing events can be queried using common tooling.

We note that *.web* archives contain all observed events that occurred during WebREC's execution of the page and not a subset of behaviors specified at *measurement time*. This has the cost of making *.web* larger than needed for the immediate measurement task but brings two corresponding benefits.

The first benefit of WebREC producing *comprehensive* recordings of page behavior is that *.web* archives are ideal for long-term, cross-purpose archiving. These rich and comprehensive archives allow researchers to ask a wide range of questions about historical data, including questions that might have been unanticipated at *archiving time*. We note that this makes *.web* archives very different from the existing archive formats discussed in Section 2.3, where datasets are generally narrowly focused on a particular research question.

And second, *.web* is *general*, allowing it to generate archives to support reproducible studies beyond what existing common measurement tools and archive formats allow. Consider the demonstrative "mixed content" example from earlier, for instance. Because WebREC directly records how the browser behaved when executing a page, it allows future researchers to accurately measure how the website (including mixed content) was experienced by past users. This is very different from existing archive formats, which record *what the browser fetched* instead of *how the browser behaved*. Only knowing what resources were fetched for past websites hinders historical accuracy and reproducibility by making it difficult-to-impossible to know if differing results are due to *endogenous differences in measurement correctness* (e.g., detecting an error in the original measurement's approach like in Section 4.4.2) or *exogenous differences in page re-execution* (e.g., browser non-determinism as discussed in Section 2.4).

In summary, WebREC is designed to be a tool for *general*, *accurate*, and *comprehensive* Web security and privacy measurements, and produces *.web* archives designed to enable accurate, reusable, general data archiving to aid reproducibility of Web measurement.

3.2 WebREC Design

We implement WebREC as a modified Chromium browser. As explained in Section 2.2.2, this abstraction level is needed to ensure *accurate*, high-fidelity insights into website executions.Chromium's market share of over 70% [64], makes it a very *general* and long-term solution. Naturally, this decision comes with the limitation that WebREC is browser-dependent. However, we do not focus on cross-browser differences but rather on capturing *accurate* and *comprehensive* Web behavior. Alternatively, supporting all major browsers would add significant complexity and maintenance challenges.

WebREC consists of three parts: i. a system for determining *which* page(s) is measured, ii. a system for measuring *what* resources are fetched during page execution, and iii. a system for measuring *how* the browser behaved while executing the page. Each capability is briefly described below.

First, a researcher directs WebREC to measure a webpage in one of two ways: either first, by using the existing, well-known, and previously discussed Puppeteer library to pragmatically navigate WebREC to a page to measure, or second, by using WebREC in an interactive session, where the researcher uses the browser as a user typically would, but with WebREC recording all relevant page behaviors.

Second, WebREC measures what resources are fetched during page execution using Chromium's built-in *DevTools* protocol, the same protocol used by Chromium's debugging tools and Puppeteer to automate Chromium instances.

Third, WebREC records browser behavior during page execution by instrumenting Blink, the open-source system used for parsing HTML, presenting Web pages, and handling the interaction between JavaScript and browser APIs used in Chromium. Our approach builds on an existing Chromium-based measurement system called PageGraph, which has been used in prior work [9, 58, 59]. PageGraph *comprehensively* tracks and attributes all network requests, DOM changes, and denoted WebAPI and JS-builtin calls, building an internal graph of annotated *actors*, *actees*, and *actions*. WebREC extends PageGraph in a number of ways, including handling and recording redirection in HTTP requests and recording the arguments to (and structured responses from) WebAPIs, among other changes needed to support the high-fidelity and retrospective use cases WebREC aims to target.

Finally, we note several attributes of WebREC's construction that better ensure accuracy and help to avoid some of the correctness issues that affect some ad-hoc, or per-project, measurement tools. For example, WebREC is validated against the cross-browser Web Platform Tests [80] to ensure stability and coverage (i.e., that WebREC can handle and record the enormous range of browser behaviors used by modern websites). Additionally, WebREC is kept current with every major Chromium release, ensuring that it accurately records the Web as executed in a then-current browser (see Section 6.5).

In short, WebREC is built to minimize (or avoid) the limita-

tions of other browser-modification Web measurement tools.

3.3 *.web* Archives

Finally, we discuss the format and contents of the *.web* files WebREC produces when measuring a page's execution.

A *.web* archive consists of three files. First, each it contains a screenshot of the measured website at the end of the measurement period (i.e., when WebREC stops measuring page behaviors and begins serializing the behaviors it recorded into a *.web* archive). This screenshot is captured using the Puppeteer library, similar to other Web measurement tools.

Second, each *.web* archive includes a copy of resources fetched during the page's execution. This archive is also collected using Puppeteer and is encoded as a HAR archive, again, in a manner similar to existing measurement tools.

Third, and most novel, each *.web* archive includes a description of all page behaviors measured during the page's execution. Rather than using existing archiving formats like HAR or WARC, we adopted PageGraph's graph-based format to represent element dependencies and action flows as a directed multi-graph (as GraphML XML). Compared to list-like formats used in traditional archives, this approach offers a more intuitive representation of action dependencies and flows. This graph records all *actor-action-actee* behavior tuples described in the previous section, capturing all observed communication, DOM-modification, and script behaviors (i.e., calls to significant Web APIs and JavaScript builtins) that occurred during the page's execution, detailed enough to allow the page's execution to be accurately replayed, step by step.

In conclusion, each of the three files included in each *.web* archive fulfills a different purpose to enable accurate, replicable, historical Web measurement: the screenshot file records the presentation of the archived page, the HAR file records the resources loaded, and the GraphML file records the iterative behavior, over the page's entire lifetime.

4 Accuracy Improvements

This section demonstrates how WebREC can be used and the improvements it offers compared to common HTTP archives, e.g., enhancing accuracy in reproducing prior experiments.

To do so, we revisit and reproduce three representative Web measurement experiments from previous works, creating WARC, HAR, and *.web* datasets for comparison. We focus on three commonly measured resources: First, we analyze the accuracy of recording *JavaScript (JS) executions* by replicating experiments from Snyder et al. [60]. Then, we compare accuracy in recording *Document Object Model (DOM)* content, a commonly measured resource, e.g., to analyze third-party roadblocks through inline event handlers for Content Security Policies (CSP) [66]. Lastly, we measure how the *HTTP requests* differ and check how this helps the reproducibility of privacy research on third parties and trackers [15]. Before the experiments begin, we need a robust measurement pipeline.

4.1 Dataset Construction

To perform meaningful analysis and comparison between different archiving formats, we need a dataset construction pipeline that meets several criteria. First, we need a *baseline* (BL) of recorded actions to compare all archives against. Otherwise, we cannot tell which archive is correct if two of them differ. Second, we require *consistent* recording of the same events during a single visit to account for non-determinism [52]. Third, the dataset needs to be *representative* for common Web security and privacy measurement studies. To meet these criteria, we built a pipeline that creates a BL dataset from requests sent to the sites from the CrUX top 10k bucket, a highly accurate and literature-recommended website popularity list [53]. We proxy all requests through commonly used archiving tools to record the same requests (see Figure 1 for the pipeline and Section 7 for ethical considerations).

Our pipeline begins with an origins list (CrUX, December 2023). For each origin, it initiates a new instance of warcprox [24] (3), and mitmdump [41] (2), and then runs WebREC (1). We chose warcprox (from the Internet Archive) and mitmdump (part of mitmproxy) because they are well-maintained and tested tools. WebREC's requests are first proxied through mitmdump and from there through warcprox to the Web server (ignoring TLS certificate errors). This setup ensures that all parts receive the same requests and responses, allowing us to record *.web*, HAR, and WARC files accurately.

With the *.web*, HAR, and WARC files, we can now compare requests and different file contents. However, it would not be possible to compare dynamic JavaScript executions on a page for HAR and WARC files. To measure executions on archived content, prior work has made use of sites like the Internet Archive that replay recorded responses [34, 51, 67]. We apply the same method by using the replay functions of mitmdump for HAR files (5) and pywb [74] (recommended by warcprox) for WARC files (6), and then measure executions by hooking into monitored JavaScript functions, keeping WebREC as unmodified as possible. The hooks are set via a prepended script element in front of every HTML response with the mitm proxy (2 & 5). Due to the error tolerance of browsers [21], this script is executed even though it is positioned in front of any HTML tag. We note that this might lead to a difference in parsing due to the now entered quirks mode [39], a point that we discuss in Section 4.2. Additionally, the script also hooks into programmatically created iframes. To ensure that the script is always executed and not blocked by CSP, we use the Puppeteer option *setBypassCSP* to deactivate CSP. This approach maintains consistency by using the same tools for the initial crawl and the replays, differing only in the response source; archive files instead of the live web.

For our baseline dataset, we use the JavaScript (2) recorded

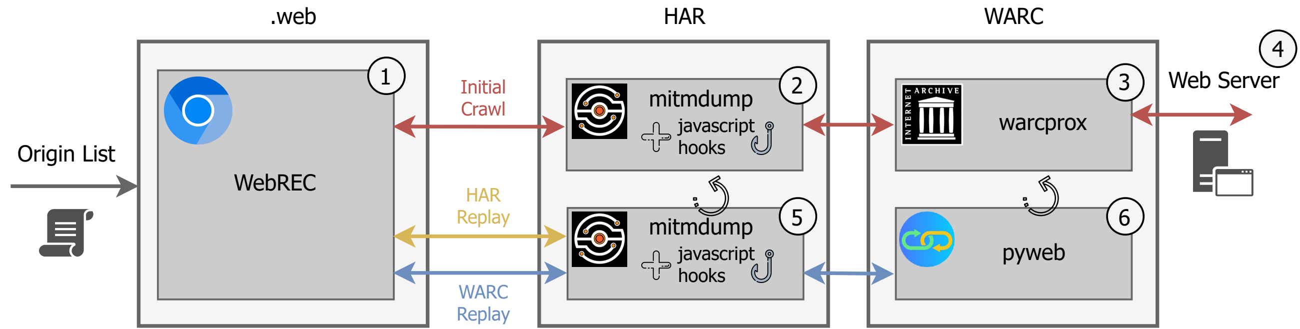

Figure 1: The pipeline used to create our inital dataset plus datasets for HAR replays and WARC replays.

with hooks during the crawl. Since WebREC is designed to avoid altering browser behavior, the JavaScript hooks remain unaffected. After completing the pipeline for the CrUX top 10k, we have a solid dataset and baseline for analysis.

4.2 Limitations

Before focusing on the analysis, we first discuss the pipeline's limitations. As previously mentioned, adding a script tag at the beginning of each document causes the browser to enter quirks mode [39]. The quirks mode is an approach by browser vendors to ensure backwards compatibility and render even severely malformed HTML with their Web browser. It causes the browser to follow other parsing algorithms and leads to slight differences in the parsed results compared to what typical users would see (e.g., CSS parsing). Yet, we are not aware of any relevant changes that would influence our results.

Furthermore, the pipeline misses executions inside of iframes added by the HTML parser if they contain a `srcdoc` property. Using a custom query with the HTTP Archive [23], we can see that only 10,534 (0.9%) of the 1,226,954 recorded iframes in their 1M dataset use `srcdoc`. While our pipeline could detect and rewrite these during the rewriting phase with mitmproxy, it could cause side effects introduced by HTML parsing libraries (e.g., by removing malformed elements) [21]. Hence, we believe the impact of such mutations could be more significant to our results than being unable to hook into 0.9% of the iframes and thus decided against this approach.

Finally, given the multi-tier architecture, there is a small chance for a race condition in the resource recording. This causes a slight mismatch in recorded resources between archives and the *.web* format, leading to a very small difference in reported resources (details in Section 4.5.2).

4.3 JavaScript

Numerous Web security and privacy measurements [31,61,62] analyze JavaScript execution. In this section, we want to assess the differences, if any, in measured JavaScript executions between reproducing experiments using a traditional archival format and our proposed *.web* format. To assess this, we replicate experiments from Snyder et al. [60], who investigated the prevalence if various JavaScript APIs. The authors injected JavaScript into each page to modify methods and properties to determine which API is used how often.

4.3.1 JavaScript Experiment

Our experiment focuses on APIs related to two standards: *CSS Object Model* (CSS-OM) [57] and *HTML Channel Messaging* (H-CM) [22]. Snyder et al.report that CSS-OM is often used and rarely blocked by privacy tools, whereas H-CM is very actively used but frequently blocked. This small scope allows us to highlight key differences between archiving techniques.

To count the API invocations after running our pipeline, we first analyze *.web*'s execution graphs for *js call* edges, which indicates API invocations. For the BL dataset and the replayed WARC and HAR responses, we count calls by analyzing the log files where the earlier mentioned JavaScript hooks log every JavaScript action on the page.

4.3.2 JavaScript Comparison

After running the pipeline on the CrUX 10k bucket, our dataset contained 8,479 *.web* files, each indicating that a crawl was successful. Next, we replayed the responses from HAR and WARC files. For 272 and 75 origins, respectively, replaying failed due to timeouts in reading the file (although set to a generous 60 seconds) or file interpretation errors. Due to overlapping issues, this leaves us with 8,150 successfully crawled and replayed origins for the remainder of this section.

For each origin, we counted the appearances of CSS-OM and H-CM API invocations. We define an *appearance* on an origin as the combination of the API name and the number of times it was executed. Our BL dataset shows 15,775,881 API calls grouped to 42,179 appearances across origins. Table 1 shows that out of these 42,179 appearances the *.web* dataset aligns with 39,098 (93%) appearances on the same number of API calls, i.e., WebREC recorded the same number

Appearances	.web		HAR		WARC	
Equal	39,603	93%	22,688	54%	22,645	54%
More	2,918	7%	1,934	5%	1,725	4%
Fewer	163	0.3%	17,557	37%	17,809	42%

Table 1: Appearances on the archived sites compared to 42,179 appearances on the baseline (BL).

```
1 <svg id="s" width="100" height="100">
2   <circle cx="50" cy="50" stroke="green" />
3 </svg>
4 <script>console.log(document.getElementById('s').style)</script>
```

Listing 1: DevTools log a CSSStyleDeclaration.

(a) Ad from live website.

(b) Ad from archived website.

Figure 3: Google ads load dynamic content leading to a difference of JS executions between live and archived websites.

of invocations as the BL. For 2,918 (7%) of the appearances, *.web* captured more API calls, which can be attributed to WebREC's deeper hooking capabilities compared to the hooks implemented in JavaScript capturing only surface-level activities. For instance, the code in Listing 1 shows a snippet in which the SVG's *CSS Style Declaration* is logged, leading to the log output in Figure 2. The lookup for attributes that appear in the log, e.g., *accentColor* and *additiveSymbols*, are also recorded in *.web* while the JavaScript hooks for the BL ignore it. Again, this difference is expected due to the underlying hooking differences. Moreover, only 163 (0.3%) appearances had fewer invocations in *.web* than in the BL, showing that researchers can achieve a nearly identical representation of API invocations as in the original visit, even in replication experiments conducted years later. Across all origins, the average difference in appearances between *.web* and BL is only 0.7%, highlighting the approach's accuracy.

After comparing the baseline to *.web*, we now focus on the comparison with replayed responses. Given that the underlying method to record the API calls is the same as for the BL, one would naturally expect results closer to the BL. The data shows that only 22,688 (54%) appearances in the HAR replays and 22,645 (54%) for WARC align with the BL, i.e., the overlap is significantly lower than for *.web* (93%). On the one hand, both HAR and WARC also have cases in which more API invocations are detected compared to the BL. We attribute this to the fact that replayed responses have no network delay when loading the resources, triggering some reoccurring API calls earlier.

On the other hand, the vast majority of appearance differences are because *fewer* invocations are detected in HAR and WARC; 17,557 (37%) and 17,809 (42%), respectively. Here, various of the reasons mentioned in Section 2.4 could be at play. For instance, our dataset reveals that sites that use Cloudflare (i.e., challenges.cloudflare.com) for bot protection tend to have fewer API calls for the replayed responses, likely due to the lack of real-time interactions with their servers, which are crucial for dynamic content exchange and behavior measurement. Another example is shown in Figure 3, a Google ad element, first on the live visited website (3a) and then on the archived website (3b). While the archived version shows only a static logo, the live version loads an advertisement after a few seconds from the ad servers leading to various CSS-related calls. Especially for privacy related research, this difference could be relevant. And this difference in appearances is significant: While we observed an average difference across all origins of 0.7% between *.web* and BL, *we measured 13.9% for HAR and 13.3% for WARC replayed responses*. This highlights the drawback of replaying content rather than recording and inspecting the executions directly.

Figure 2: The DevTool logs show attributes of a CSSStyleDeclaration that are also logged with WebREC.

4.4 Document Object Model

Another typically analyzed component in measurement studies is the DOM of a webpage and how it behaves, for example, in case of DOM Clobbering [30], or with invalid HTML [21]. To showcase how *.web* could be used for such experiments, we replicate a measurement from Steffens et al. [66], in which they analyzed to what extent third-party content on webpages complicates the security mechanisms Content Security Policy (CSP) and Subresource Integrity. In this section, we replicate one part of their hypothetical what-if experiment, in which the authors make the assumption that first-party developers want to deploy a CSP without using `unsafe-inline` and `unsafe-eval`. Their findings revealed that the vast majority of sites (86%) are unable to deploy CSP due to JavaScript code, which adds inline event handlers to the DOM requiring `unsafe-inline`; we, therefore, focus in this experiment on

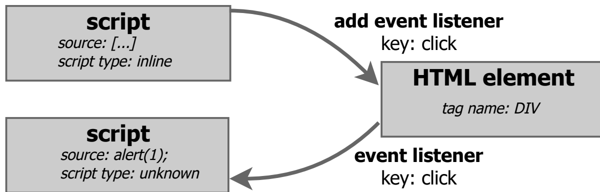

Figure 4: One example how an EventListener is represented as script node in the execution graph.

JavaScript-added inline event handlers.

4.4.1 DOM Experiment

To analyze inline event handlers, we run Steffens et al.'s open-sourced SMURF code in our pipeline to count all origins that use inline event handlers added to DOM through APIs like `document.write` and `innerHTML`. Using this code, we can simply count the origins in our log files and use it as a baseline for the comparison. We use the same method for the replayed responses from HAR and WARC files. Within *.web*, we query the graph for *event listener* edges (see Figure 4). If these point to the same JavaScript node that added the event listener (i.e., *add event listener* edge) or the script node does not have the type *unknown*, they are *programatically added*, i.e., do not involve string-to-code transformation. If, instead, they were added from the HTML parser or another script, they are actual inline event handlers, which would require `unsafe-inline`.

4.4.2 DOM Comparison

In this crawl, we successfully visited 8,523 origins with 311 HAR and 44 WARC (overlapping) issues replaying responses, similar to the previous experiment. Thus, this sections builds on 8,170 visits to analysis.

The results of the analysis are shown in Table 2. At first sight, the results appear extremely surprising: the original version of Steffens et al.'s code, SMURF [65], finds four times as many sites with inline event handlers (6,188) compared to WebREC (1,515). We, therefore, carefully analyzed SMURF's source code and found a number of impactful bugs. First, their code hooks into APIs which can be used to add elements to the DOM and subsequently checks if the added element contains inline event handlers. However, their code iterates over all the *properties* of an element instead of checking for *attributes*. This difference is critical, though: if an event handler is programmatically added through a function reference, this is a *property* of the element but not an *attribute*. In contrast, adding an inline event handler through `ele.setAttribute('onerror', '...')` (which requires string-to-code transformation) sets the attribute. A programmatically added event listener for an HTML node is a common

Method	SMURF (bug)	SMURF (fix)	*.web*	WARC	HAR
Origins	6,188	1,644	1,515	1,472	1,368

Table 2: Origins for which we found inline event handlers.

way for many third parties to react to events while still allowing CSP to block inline scripts (a real-world example can be found in Appendix A). The SMURF code also does not properly capture event handlers which are nested inside a subtree of the DOM, leading to false negatives. Finally, SMURF assumes that any attribute starting with *on* is an event handler; as a result, our initial experiments showed that due to typos (e.g., *onclik*), HTML elements without meaningful event handlers were flagged as CSP blocking. Upon discovering of these three issues, we contacted the original authors and fixed the code together with them to have a more meaningful baseline.

Given the updated code, we now investigate the feasibility of WARC, HAR, and *.web* to a study such as Steffens et al.. We find that the replayed responses from WARC and HAR show fewer origins with inline event handlers, 1,472 and 1,368 respectively, than the initial baseline of 1,644 origins. This discrepancy is again due to dynamic elements, such as JavaScript, not reloading accurately. These findings reinforce our argument that archive-with-metadata approaches are not ideal when the goal is to accurately reproduce Web measurements.

For WebREC, there remains a difference from the baseline (1,644 vs. 1,515). Closer analysis of the findings showed that one issue relates to PageGraph, which was not able to handle event handlers inside SVG elements. We reported this limitation to the authors who addressed it in a newer version. Moreover, SMURF just logs any event handler regardless of whether it was added to the page's DOM and triggered (media elements are an exception) and irrespective of their origin, i.e., also reports those added in cross-origin iframes, which are not relevant for CSP. Hence, we believe the fixed implementation of SMURF to be in line with what WebREC can record.

4.4.3 Consequences

In order to conclusively refute the findings of Steffens et al., we would have to apply the fixed version of their code to the dataset from 2020. However, without a published dataset that is accurately reproducible like WebREC, we cannot make ultimate statements; even if the authors had recorded HAR or WARC files, it would be infeasible to fully replicate their findings. Nevertheless, we believe that inline event handlers played a less significant role for CSP blockage (assuming the Web has not radically changed since 2020). However, the authors also crawled sub-pages that could contain more event handlers and analyzed other aspects like inline scripts or eval calls, which we did not investigate. This makes us believe that the general takeaway of the paper remains valid.

In general, our findings show that the complexity of Web

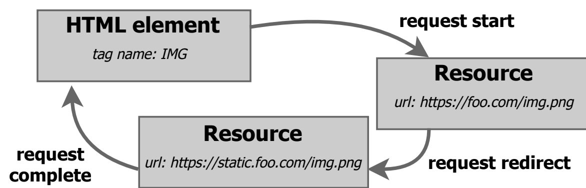

Figure 5: Resource requests and redirects in the execution graph are represented as a resource nodes with edges.

measurements and JavaScript hooking is prone to cause errors. Additionally, mistakenly collecting all *on** attributes or missing newly introduced event listeners cause problems in such traditional Web experiments. As WebREC records all activities on a webpage, the list of event listeners is always accurate. Consequently, we believe researchers utilizing this technology would significantly improve their experiment pipeline by reducing their chance of implementing bugs.

4.5 Requested Resources

Requests are frequently analyzed in Web security and privacy studies for various reasons, for example, to support arguments about security headers [7], to identify privacy leakages [56], or to measure online advertisement and tracking behavior [15, 34]. For this section, we take the online tracking paper by Englehardt and Narayanan [15], a study in which the authors captured requests from one million websites and utilized the two tracking-protection lists Easylist and EasyPrivacy [14] to measure the prevalence of third-party requests and online trackers. Following their method, we can assess how well we can perform such an experiment using WebREC and show what additional benefits researchers could get from it.

4.5.1 Resources Experiment

Unlike the experiments before, researchers would typically not replay responses since all requests are stored in the HAR or WARC files already. Hence, we follow this method and only compare *.web* against HAR in this section without replaying responses. Additionally, we want to demonstrate how comprehensive WebREC's recorded page behavior is and use only this feature from *.web* to compare against HAR.

In these behavior recordings, we have two types of requests. Requests directed to load documents are tracked as URLs in the document node, i.e., the HTML pages themselves or any embedded document. Besides document requests, WebREC records resource requests and represents them via an edge to a resource node as demonstrated in Figure 5.

4.5.2 Correctness

Our results show that the requests in *.web*'s behavior records have great agreement with those recorded in HAR. In total, we count 778,500 requests in HAR after filtering out non-page-related requests like service workers or initial redirects, while we see 776,229 requests captured via WebREC for 8,544 successfully crawled origins. This makes a negligible difference of less than 0.3%.

Filtering the HAR file is required due to the very different ways these two file formats work. HAR is developed to record *all* requests that leave the browser, while WebREC captures every activity happening *on the page*. These do not include activities like service workers, pre-flight requests, WebSocket upgrades, or CSP reports. Also, initial redirects are out of scope as they do not belong to the *page*.

We can attribute most requests made by the browser and filter them out from the HAR dataset by relying primarily on request headers. We attributed 12,342 pre-flight requests looking for the `Access-Control-Request-Method` header and 1,210 CSP report requests using `Sec-Fetch-Dest` headers set to `report`. WebREC behaviors do also not record the upgrade request for a WebSocket which we attribute via the `Upgrade: websocket` header, leading to 1,455 requests being filtered out. To attribute initial redirects from a document, e.g., *mobile.example.com* to *example.com*, we used the Sec-Fetch-Dest `document` and the `Location` response headers which gave us 2,819 filtered requests. Additionally, Chromium itself causes more than 40 requests per session, e.g., requests to the updater and similar, leading to 449,511 requests, which we attribute by looking for these URLs. Lastly, service workers also live in their own context and are not recorded in *.web* behaviors. While the initial loading of a service worker is detectable through the Sec-Fetch-Dest header, scripts included by the service worker do not carry such distinctive features. Therefore, we instead hooked into the serviceWorker API to disallow registration of any service workers.

Besides requests outside the page context, we also see requests that are internally made by the page but that never leave the browser. These never touch the HAR proxy but are recorded via WebREC. For example, some `Non-Authoritative Information` responses, like HSTS upgrades, are internally represented as `307 Internal Redirect` or the WebRequestAPI, likely Chromium internal behavior. Lastly, our manual investigation into differing requests shows a small race condition, a limitation in our pipeline: Sometimes, the browser sends a request just before we generate the *.web* output and tear down our pipeline. This means, a response might not return in time, resulting in an unrecorded HAR entry but an existing request in the behavior recording. Alternatively, the request might return and be recorded in HAR, but only after the *.web* is generated, leading to one more request in HAR. Due to these racy requests and the page-internal ones we could not to filter out, we measure

a difference of less than 0.3% in the total number of requests.

This comparison shows that WebREC is well-suited for measuring activities directly *on the page*, while the proxy-generated HAR captures *all* requests – information, often not needed for many studies, as the next section shows.

4.5.3 Replication

This section replicates a common third-party experiment to demonstrate that the page-context-only approach produces nearly the same outcomes as the more inflated HAR.

In the more than 700,000 requests we made to the top 10K websites, we see that 5,938 third parties are present on at least two first parties in the *.web* behavior records. At the same time, in HAR recordings, after filtering out the document redirects to avoid false positives (i.e., the false origin), we see 6,043 third-party requests. Same as for previous work, we can see that the majority of third parties are only present on a few sites as only 719 (*.web*) and 739 (HAR) third parties are present on more than 1% of all sites in our dataset. Next, we measured the prevalence of the third parties and compared the results to tracking-protection lists. The results for the top 20 third parties are demonstrated in Figure 6, showing the same observations as made by Englehardt and Narayanan [15]. As explained in their paper, the top third parties belong to only a small group of organizations. Indeed, the top 8 alone are associated with Google. Additionally, not all third parties are used in a tracking context, such as content delivery websites like *gstatic.com* or *jsdeliver.net*. The data confirms that whether measuring requests from the HAR recordings or utilizing activities captured with WebREC, the overall picture and takeaways remain consistent since the original study.

4.5.4 Attribution

In some experiments, simply identifying the request-target might not be enough. For example, if we want to understand the supply chain of third-party dependencies, we need to properly attribute the initiator of a request. However, it is non-trivial to accurately attribute requests to the corresponding source from which they were requested if the measurement is only based on recorded requests. Of course, indicators like the Referer header offer some insights, yet the Referrer-Policy could skew these signals. Consequently, a thorough understanding of third-party requests necessitates insights into on-page activities, activities recorded with WebREC.

We now demonstrate the additional insights that *.web* can provide compared to traditional HTTP archives. To that end, we analyze the requests sent towards the most prevalent third party, i.e., Google Tag Manager (GTM), which received a total of 12,564 requests from 4,642 origins.

Table 3 shows different variants through which the GTM is requested. Unsurprisingly, most requests come from script elements adding GTM. The most responsible party (causing

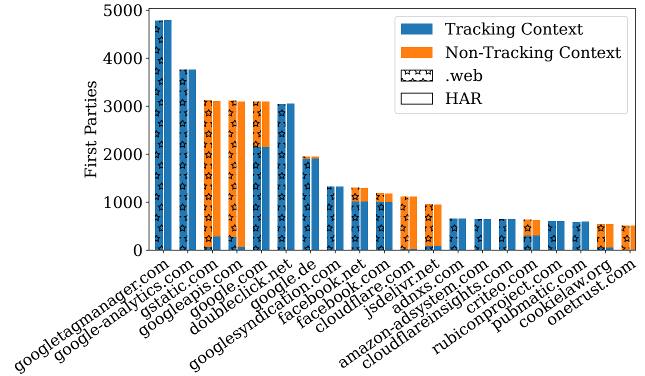

Figure 6: Top third parties on the top 10K sites via HAR and *.web*, divided in tracking and non-tracking context.

5,130 requests) is GTM itself, as the initial GTM script often includes additional scripts via script elements. It is followed by the script elements added by the first party, the expected way to add GTM in the first place. The long tail of third parties responsible for script elements (736 different third parties; not shown in the table), demonstrates the wide variety of entities websites use to load JavaScript modules, including alternative tag managers like Adobe Dynamic Tag Management or own content delivery network such as *xhcdn.com*.

GTM is also used to load images (e.g., tracking pixels), mostly requested by GTM itself (563). Requests made through JavaScript fetch or XMLHttpRequests APIs towards GTM are mainly caused by the first party (157). In addition, 92 requests were caused by parallel-loaded script elements (e.g., *defer*) within the page's original HTML (*Parser*).

This shows that WebREC's behavior recordings can be used for both to accurately measure requests and to further understand why and how these requests occurred by providing far more details than a basic request record does, which would have to rely on heuristics to determine this metadata.

5 Generality of WebREC

Previous sections showed that common Web measurement methods lack reproducibility and sometimes correctness. We proposed WebREC as a general-purpose tool to overcome these issues, yet its generalizability remains in question.

To evaluate generality, we surveyed previous Web measurement studies from top security and privacy conferences, building on a SoK paper by Stafeev and Pellegrino [63], who analyzed the use of crawlers in Web measurement studies. As detailed in Appendix B we consider a subset of their reviewed papers, including studies from 2020 to 2022 that measured data within the scope of what WebREC offers, i.e., they either analyze webpage JavaScript executions, the page's content, or HTTP requests made. Thus, we focus on a set of 97 papers.

Our survey shows encouraging numbers, suggesting that 68

Context	Responsible Party	# Req.	# Origins
Script Element	*googletagmanager.com*	5,130	2,751
	first-party	4,342	3,503
	google-analytics.com	593	445
	adobedtm.com	77	40
	xhcdn.com	60	60
	...	1,534	1,169
IMG Element	*googletagmanager.com*	563	96
	halodoc.com	8	1
	googleoptimize.com	1	1
Fetch/XHR	*first-party*	157	1151
	sports.ru	2	2
	thesimsresource.com	2	1
	geekdo-static.om	1	1
Parser	*first-party*	92	74

Table 3: Third party requests to GTM grouped by context and responsible party who sent the request.

(70%) of the 97 papers could use WebREC for most of their data collection. This would not only standardize the process and reduce the chance for errors during collection but also save researchers resources and time. For example, Musch and Johns [43] investigated JavaScript anti-debugging techniques in the wild by identifying code patterns and assessing performance changes, a tremendous development effort. Since these techniques are registered during page load, *.web* contains them, allowing this analysis without a custom crawler.

On the other side, studies unsuitable for WebREC include those measuring something outside the page context, e.g., service workers [29] or browser extensions and those focusing on very specific features, such as DNS over QUIC [32]. Additionally, since WebREC is based on Chromium, studies requiring other engines like Firefox are not catered for [44].

Even though 70% of papers could rely on WebREC instead of a custom crawler, the researchers would still require crawling, leading to avoidable resource usage and comparability issues. In fact, our investigation also shows that 47 (48%) papers could base their analysis solely on a *.web* archive instead, without the need for them to run a crawler. For example, Luo et al. [36] conducted an experiment labeling DOM elements as sensitive, subsequently checking what third-party scripts access them. Calzavara et al. [8] studied inconsistencies in security mechanisms comparing cookie policies and security headers across sites. Both DOMs and headers are collected by WebREC and could be analyzed in a *.web* archive.

To conclude, we believe that a significant percentage of Web measurement studies (70%) could benefit from using WebREC over custom crawlers and that 48% could even be based solely on a *.web* archive. This highlights WebREC's potential to streamline measurement processes and to enhance reproducibility and correctness in our academic field.

6 Discussion

With the results of the previous section in mind, we now summarize the primary benefits of WebREC.

6.1 Reproducibility

WebREC addresses the problem of reproducibility by providing a comprehensive bundle of information about page requests and activities for researchers to use and share. Recorded at the core of the browser, this information offers a much more accurate representation of page behavior than existing archiving formats. For instance, for JavaScript API calls, WebREC shows an average difference of only 0.7% from the baseline, in contrast to a 13% difference with common HTTP archives. This discrepancy arises from dynamic behaviors like live server interactions, which can never be captured accurately by simply replaying responses. WebREC eliminates this need for replaying responses by storing all relevant activity information at runtime. Researchers can reproduce experiments by running the same scripts used in the original study on their dataset, ensuring reliable replication.

Going one step further than reproducing experiments, *.web* also allows researchers to delve deeper into the initially recorded data, offering a better understanding of various aspects of a historical experiment. We demonstrated for the requested resources experiment that we can not only replicate the key findings of the original paper but we also easily attributed third-party calls to different scripts, providing valuable insights into third-party deployment. This advanced capability can help researchers explain why replicated findings might differ over time, e.g., due to changes in the mixed content policy. We encourage researchers to provide their measured data as *.web* archives for future research to use.

6.2 Correctness

Since WebREC records behavior at the browser's core, the recorded behavior is correct and eliminates the need for researchers to modify browsers or inject code into Web pages.

Custom modifications can introduce issues if researchers are not fully aware of every browser-specific detail, leading to unintended consequences. For example, we showed, how a study [66] mistakenly attributed programmatically added event handlers as inline ones due to a mix-up between *properties* and *attributes*. WebREC avoids such problems by recording all page activities accurately at its core, allowing researchers to analyze *.web* without an in-depth understanding of the HTML or other specifications. In *.web*'s execution graph, the source of an event handler is clearly visible, reducing the risk of misattribution during analysis significantly.

Even if a potential bug occurs in the analysis phase, other researchers could easily reconstruct and correct it based on the

provided *.web* bundles. That this is desirable shows the Steffens et al. [66] paper, for which we could not rerun their experiment with our corrected script due to the lack of an available dataset. Thus, we only run their script on our smaller dataset years later showing that over 50% of the visited sites in our data would have incorrectly attributed event handlers. How this would change the takeaway message of the original paper is unclear. We believe providing a format that allows other researchers to rerun and even correct experiments significantly improves the correctness of Web measurement research.

6.3 Efficiency

WebREC improves the efficiency of Web measurement research. The experiments in Section 4 demonstrate how *.web* supports various studies, eliminating the need for resource-intensive development of custom crawlers. Around 70% of papers we reviewed could use WebREC, highlighting the benefits of a standardized crawler for our community. On top of that, 48% of these papers conduct experiments that could rely solely on *.web* archives without the need to live-crawl. To make this possible for the community, we plan a publicly available, actively maintained *.web* archive for future researchers (see Section 7). We are confident that this will save resources, reduce Web traffic, and prevent potential ethical pitfalls.

Such an archive would allow extensive future work, like longitudinal JavaScript behavior analysis, currently not feasible with traditional Web archives. It also allows easy prototyping of experiments without the need for additional crawls, thereby improving efficiency in Web measurement research.

6.4 Storage Implications

Storing more information is expected to result in larger file sizes, requiring researchers to balance the new insights gained with storage demands. To address this, we added an optional compression step to our crawler and query tool. To estimate the expected storage requirements, we conducted another experiment comparing *.web* with the HAR files generated by a proxy. We ran our pipeline without any JavaScript injections (which would otherwise inflate the required space) on the CrUX 10k list, ending with 8,934 origins to compare.

For these origins, the average HAR file size was 13.6 MB, while *.web* files averaged 10.2 MB, broken down as 8.2 MB for HAR files, 1.5 MB for behavior records, and 0.4 MB for screenshots. The difference in HAR and *.web* HAR sizes is due to the fact that proxy-created HAR files include all browser requests, such as upgrade requests or updating dictionary files, whereas the *.web* HAR file only captures requests originating from the page context. This shows that WebREC, while providing more insights, keeps storage needs low.

6.5 Maintenance Overhead

WebREC is developed as a series of modifications to "stock" Chromium, along with external tools to interact with those modifications. Maintaining modifications in a rapidly changing code base like Chromium is a challenge, which can lead to quickly-abandoned prototypes, especially in research. To address this WebREC is designed as an ongoing solution, with design choices that dramatically reduce the maintenance overhead in keeping WebREC current.

Most of WebREC's complexity comes from modifications to Chromium and its sub-projects (e.g., Blink, V8), which are mostly implemented in C++. The frequent changes to the code base make correctly maintaining modifications outside the project difficult. For example, patches that work correctly against one version of Chromium can need significant reworking for the next version, if Chromium developers have refactored code or added significant new functionality.

WebREC is implemented using strategies that simplify updates to new Chromium versions, and minimize amount of patching needed. First, most of WebREC's logic is implemented separately and independently of upstream Chromium code, kept in its own C++ namespace and data structures (instead of modifying the ones relied on by Chromium). Second, WebREC leverages many of Chromium's systems to capture information about a page's execution without needing to modify Chromium code. For example, Chromium's *CoreProbe* system allows code to receive notifications when certain page activities occur (e.g., new elements created). Third, Chromium's build system and C++ classes include mechanisms to augment Chromium's classes in a more robust way than sub-classing. For example, the classes `Supplementable` and `Supplement` allow developers to add behavior to existing "Supplementable" classes, by implementing that new functionality in a "peer Supplement" class, without needing subclasses or changing upstream code, further making WebREC's behavior robust across Chromium changes. And fourth, when the previously discussed approaches are not possible, WebREC uses subclasses when possible, and modifies Chromium code through clever use of C/C++'s pre-processor (instead of through patches), both of which, while not ideal, end up being easier to carry across Chromium versions than patches. All this allows us to maintain WebREC with only 33 patches (compared with over 12k lines to implement WebREC's browser-side functionality.

This design has been very successful in making WebREC easy to maintain across Chromium releases. WebREC has been kept current with every Chromium version since September 2022, (i.e., 40 Chromium versions) with little to no changes required each update. We estimate less than an hour of time has been needed to keep WebREC current for each Chromium release, many needing no additional effort at all.

7 Conclusion

This paper highlights a significant issue in the current Web measurement landscape: the absence of general-purpose measurement tools and archives. Instead, we see self-implemented, single-use crawlers. That this approach easily leads to correctness and reproducibility issues is shown by a case study where we identified errors in a published paper misattributing event handlers. For Web archives, we find over 13% discrepancy in JavaScript API call appearances between replayed archives and our baseline, revealing accuracy gaps.

To tackle these issues and achieve reproducible, efficient, and accurate Web measurements, we then presented and evaluated WebREC. It bundles a Web page's HTTP communication, an execution graph, and screenshots into a *.web* archive, proving more accurate in replicating page behavior than traditional archives. Many existing measurement studies could have benefited from using it, offering a solution to current problems and improving future measurement research.

Acknowledgments

We thank the USENIX reviewers for their valuable comments and suggestions, which contributed to improving the presentation of our paper. We are particularly grateful to our shepherd for their guidance and advice. Additionally, we want to thank Jannis Rautenstrauch for his HTTPArchive skills, paper tips, and proofreading.

Further, we want to thank several people who have done invaluable work on the PageGraph project at Brave Software, including Anton Lazarev, Brian Johnson, and, in particular, Aleksei Khoroshilov. Lastly, we want to appreciate the work on PageGraph done by Michael Smith (University of California; San Diego) during an internship at Brave.

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

Ethical Considerations

As USENIX Security rightfully states, ethics plays a critical role in conducting research. Therefore, we carefully considered our project's ethical implications before we started our research. In our study, we sent GET requests to the 10k most popular origins according to the CrUX list [53]. This is a research practice employed by numerous Web studies before and is generally considered ethical in the security and privacy community. However, it is essential to acknowledge that ethical standards can evolve, and each project should be assessed on a case-by-case basis.

In this study, we requested each origin of the top 10k only once per experiment with a standard GET request. Since a GET request is designed to receive data without making any modifications, we assume no negative impact for the end user. We also ensure no negative impact for the website operators as our minimal interaction does not contribute significantly to the common internet traffic or pose any more risk than typical internet background noise. While we, however, might potentially violate some websites' terms of service, as we did not verify whether automated site visits were explicitly allowed, we believe that the benefits to the research community in the form of our measurements and the demonstration of WebREC outweigh this concern. All our additional measurements are performed on our client side, i.e., our server, not influencing any third-party server. In conclusion, we are confident that our work did not cause harm to any individual or organization. Our research and measurements aim to contribute benefit to our research community by providing and influencing new ways how to conduct future Web measurement studies.

Open Science

For future Web measurement research, we keep our modified Chromium browser up to date and make it public for others to use [6]. WebREC is further actively developed as an open-source crawler under the name pagegraph-crawl [4]. Additionally, we provide a query tool that allows querying the produced format [5]. We plan to regularly update a collection of the top 10k websites collected in the WebREC format.

To promote transparency and reproducibility in our research field, we publish the code we used in this paper along with this publication [19]. This includes the pipeline to create our datasets and the scripts used to analyze the results. In addition, we have uploaded the top 1k entries of our datasets to Zenodo [18]. Due to its size, the full dataset is available upon request.

References

[1] Syed Suleman Ahmad, Muhammad Daniyal Dar, Muhammad Fareed Zaffar, Narseo Vallina-Rodriguez, and Rishab Nithyanand. Apophanies or Epiphanies? How Crawlers Impact Our Understanding of the Web. In *Proceedings of The Web Conference 2020*, 2020. doi:10.1145/3366423.3380113.

[2] Lujo Bauer and Giancarlo Pellegrino. USENIX Security '25 Call for Papers. https://www.usenix.org/conference/usenixsecurity25/call-for-papers.

[3] Dino Bollinger, Karel Kubicek, Carlos Cotrini, and David Basin. Automating cookie consent and GDPR violation detection. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/bollinger.

[4] Brave Software. pagegraph-crawl. https://github.com/brave/pagegraph-crawl.

[5] Brave Software. pagegraph-query. https://github.com/brave-experiments/pagegraph-query.

[6] Brave Software. PageGraph. https://github.com/brave/brave-browser/wiki/PageGraph, 2023.

[7] Stefano Calzavara, Sebastian Roth, Alvise Rabitti, Michael Backes, and Ben Stock. A Tale of Two Headers: A Formal Analysis of Inconsistent Click-Jacking Protection on the Web. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/calzavara.

[8] Stefano Calzavara, Tobias Urban, Dennis Tatang, Marius Steffens, Ben Stock, et al. Reining in the web's inconsistencies with site policy. In *Proceedings of the Network and Distributed System Security Symposium 2021*, 2021. doi:10.14722/ndss.2021.23091.

[9] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. Detecting filter list evasion with event-loop-turn granularity javascript signatures. In *2021 IEEE Symposium on Security and Privacy*, 2021. doi:10.1109/SP40001.2021.00007.

[10] Chrome DevTools. Protocol. https://chromedevtools.github.io/devtools-protocol.

[11] curl. https://curl.se/.

[12] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. In *Proceedings of the 26th Network and Distributed System Security Symposium*, 2018. doi:10.14722/ndss.2019.23378.

[13] Nurullah Demir, Matteo Große-Kampmann, Tobias Urban, Christian Wressnegger, Thorsten Holz, and Norbert Pohlmann. Reproducibility and Replicability of Web Measurement Studies. In *Proceedings of the ACM Web Conference 2022*, 2022. doi:10.1145/3485447.3512214.

[14] EasyList. Overview. https://easylist.to/.

[15] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016. doi:10.1145/2976749.2978313.

[16] GNU Project. Gnu wget. https://www.gnu.org/software/wget/.

[17] Google, Inc. Puppeteer. https://github.com/puppeteer/puppeteer.

[18] Florian Hantke. Datasets. https://doi.org/10.5281/zenodo.14760512.

[19] Florian Hantke. WebREC. https://doi.org/10.5281/zenodo.14718651.

[20] Florian Hantke, Stefano Calzavara, Moritz Wilhelm, Alvise Rabitti, and Ben Stock. You Call This Archaeology? Evaluating Web Archives for Reproducible Web Security Measurements. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023. doi:10.1145/3576915.3616688.

[21] Florian Hantke and Ben Stock. HTML Violations and Where to Find Them: A Longitudinal Analysis of Specification Violations in HTML. In *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022. doi:10.1145/3517745.3561437.

[22] HTML Living Standard. Channel messaging. https://html.spec.whatwg.org/multipage/web-messaging.html#channel-messaging.

[23] HTTP Archive. How do i use bigquery to write custom queries over the data? https://httparchive.org/faq#how-do-i-use-bigquery-to-write-custom-queries-over-the-data.

[24] Internet Archive. Warcprox - WARC writing MITM HTTP/S proxy. https://github.com/internetarchive/warcprox.

[25] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean-Michel Picod, and Elie Bursztein. Cloak of Visibility: Detecting When Machines Browse a Different Web. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016. doi:10.1109/SP.2016.50.

[26] Umar Iqbal, Peter Snyder, Shitong Zhu, Benjamin Livshits, Zhiyun Qian, and Zubair Shafiq. Adgraph: A graph-based approach to ad and tracker blocking. In *2020 IEEE Symposium on Security and Privacy*, 2020. doi:10.1109/SP40000.2020.00005.

[27] Umar Iqbal, Charlie Wolfe, Charles Nguyen, Steven Englehardt, and Zubair Shafiq. Khaleesi: Breaker of advertising and tracking request chains. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/iqbal.

[28] Jordan Jueckstock and Alexandros Kapravelos. VisibleV8: In-browser Monitoring of JavaScript in the Wild. In *Proceedings of the ACM Internet Measurement Conference (IMC)*, 2019. doi:10.1145/3355369.3355599.

[29] Soroush Karami, Panagiotis Ilia, and Jason Polakis. Awakening the web's sleeper agents: Misusing service workers for privacy leakage. In *Network and Distributed System Security Symposium*, 2021. doi:10.14722/ndss.2021.23104.

[30] Soheil Khodayari and Giancarlo Pellegrino. It's (DOM) Clobbering Time: Attack Techniques, Prevalence, and Defenses. In *44th IEEE Symposium on Security and Privacy*, 2023. doi:10.1109/SP46215.2023.10179403.

[31] David Klein, Thomas Barber, Souphiane Bensalim, Ben Stock, and Martin Johns. Hand Sanitizers in the Wild: A Large-scale Study of Custom JavaScript Sanitizer Functions. In *Proceedings of the IEEE European Symposium on Security and Privacy*, 2022. doi:10.1109/EuroSP53844.2022.00023.

[32] Mike Kosek, Luca Schumann, Robin Marx, Trinh Viet Doan, and Vaibhav Bajpai. Dns privacy with speed? evaluating dns over quic and its impact on web performance. In *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022. doi:10.1145/3517745.3561445.

[33] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. Fingerprinting in style: Detecting browser extensions via injected style sheets. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/laperdrix.

[34] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lerner.

[35] David Lie and Engin Kirda. ACM CCS 2024 Call for Papers. https://www.sigsac.org/ccs/CCS2024/call-for/call-for-papers.html.

[36] Wu Luo, Xuhua Ding, Pengfei Wu, Xiaolei Zhang, Qingni Shen, and Zhonghai Wu. Scriptchecker: To tame third-party script execution with task capabilities. In *Proceedings of the Network and Distributed System Security Symposium 2022*, 2022. doi:10.14722/ndss.2022.24382.

[37] MDN. Critical rendering path. https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path.

[38] MDN. How the web works. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works.

[39] MDN. Quirks Mode. https://developer.mozilla.org/en-US/docs/Web/HTML/Quirks_Mode_and_Standards_Mode.

[40] Roland Meier, Vincent Lenders, and Laurent Vanbever. ditto: Wan traffic obfuscation at line rate. In *Proceedings of the Network and Distributed System Security Symposium 2022*, 2022. doi:10.14722/ndss.2021.24444.

[41] Mitmproxy Project. mitmproxy docs. https://docs.mitmproxy.org/.

[42] Mozilla. gecko-dev. https://github.com/mozilla/gecko-dev.

[43] Marius Musch and Martin Johns. U Can't Debug This: Detecting JavaScript Anti-Debugging Techniques in the Wild. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/musch.

[44] Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm, Sorin Lerner, Hovav Shacham, and Deian Stefan. Retrofitting fine grain isolation in the firefox renderer. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/narayan.

[45] JC Pavur, Martin Strohmeier, Vincent Lenders, and Ivan Martinovic. Qpep: An actionable approach to secure and performant broadband from geostationary orbit. In *Proceedings of the Network and Distributed System Security Symposium 2021*, 2021. doi:10.14722/ndss.2021.24074.

[46] Stijn Pletinckx, Kevin Borgolte, and Tobias Fiebig. Out of sight, out of mind: Detecting orphaned web pages at internet-scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021. doi:10.1145/3460120.3485367.

[47] 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 Network and Distributed Systems Security Symposium 2019*, 2018. doi:10.14722/ndss.2019.23386.

[48] Christina Pöpper and Hamed Okhravi. NDSS Symposium 2025 Call for Papers. https://www.ndss-symposium.org/ndss2025/submisions/call-for-papers/.

[49] Jannis Rautenstrauch, Giancarlo Pellegrino, and Ben Stock. The Leaky Web: Automated Discovery of Cross-Site Information Leaks in Browsers and the Web. In *2023 IEEE Symposium on Security and Privacy*, 2023. doi:10.1109/SP46215.2023.10179311.

[50] Richard Roberts, Yaelle Goldschlag, Rachel Walter, Taejoong Chung, Alan Mislove, and Dave Levin. You Are Who You Appear to Be: A Longitudinal Study of Domain Impersonation in TLS Certificates. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19. ACM, 2019. doi:10.1145/3319535.3363188.

[51] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. Complex security policy? a longitudinal analysis of deployed content security policies. In *Proceedings of the 27th Network and Distributed System Security Symposium*, 2020. doi:10.14722/ndss.2020.23046.

[52] Sebastian Roth, Stefano Calzavara, Moritz Wilhelm, Alvise Rabitti, and Ben Stock. The Security Lottery: Measuring Client-Side Web Security Inconsistencies. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/roth.

[53] Kimberly Ruth, Aurore Fass, Jonathan Azose, Mark Pearson, Emma Thomas, Caitlin Sadowski, and Zakir Durumeric. A World Wide View of Browsing the World Wide Web. In *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022. doi:10.1145/3517745.3561418.

[54] Iskander Sanchez-Rola, Matteo Dell'Amico, Davide Balzarotti, Pierre-Antoine Vervier, and Leyla Bilge. Journey to the center of the cookie ecosystem: Unraveling actors' roles and relationships. In *2021 IEEE Symposium on Security and Privacy*, 2021. doi:10.1109/SP40001.2021.9796062.

[55] Selenium Project. Selenium History. https://www.selenium.dev/history/.

[56] Asuman Senol, Gunes Acar, Mathias Humbert, and Frederik Zuiderveen Borgesius. Leaky forms: A study of email and password exfiltration before form submission. In *31st USENIX Security Symposium (USENIX Security 22)*, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/senol.

[57] Anne van Kesteren Simon Pieters, Glenn Adams. Css object model (css-om). https://www.w3.org/TR/cssom-1/.

[58] Michael Smith, Pete Snyder, Benjamin Livshits, and Deian Stefan. Sugarcoat: Programmatically generating privacy-preserving, web-compatible resource replacements for content blocking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021. doi:10.1145/3460120.3484578.

[59] Michael Smith, Peter Snyder, Moritz Haller, Benjamin Livshits, Deian Stefan, and Hamed Haddadi. Blocked or Broken? Automatically Detecting When Privacy Interventions Break Websites. *Proceedings on Privacy Enhancing Technologies*, 4, 2022. doi:10.56553/popets-2022-0096.

[60] Peter Snyder, Lara Ansari, Cynthia Taylor, and Chris Kanich. Browser Feature Usage on the Modern Web. In *Proceedings of the 2016 Internet Measurement Conference*, 2016. doi:10.1145/2987443.2987466.

[61] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017. doi:10.1145/3133956.3133966.

[62] Johnny So, Michael Ferdman, and Nick Nikiforakis. The More Things Change, the More They Stay the Same: Integrity of Modern JavaScript. In *Proceedings of the ACM Web Conference 2023*, 2023. doi:10.1145/3543507.3583395.

[63] Aleksei Stafeev and Giancarlo Pellegrino. SoK: State of the Krawlers–Evaluating the Effectiveness of Crawling Algorithms for Web Security Measurements. In *33rd USENIX Security Symposium (USENIX Security 2024)*, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/stafeev.

[64] StatCounter. Browser market share worldwide. https://gs.statcounter.com/browser-market-share.

[65] Marius Steffens, Marius Musch, Martin Johns, and Ben Stock. Open-sourced version of smurf. https://smurf-ndss.github.io/.

[66] Marius Steffens, Marius Musch, Martin Johns, and Ben Stock. Whos Hosting the Block Party? Studying Third-Party Blockage of CSP and SRI. In *Proceedings 2021 Network and Distributed System Security Symposium*, 2021. doi:10.14722/ndss.2021.24028.

[67] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. How the Web Tangled Itself: Uncovering the History of Client-Side Web (In) Security. In *26th USENIX Security Symposium*

(USENIX Security 17), 2017. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/stock.

[68] Ben Stock, Stephan Pfistner, Bernd Kaiser, Sebastian Lekies, and Martin Johns. From facepalm to brain bender: Exploring client-side cross-site scripting. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015. doi:10.1145/2810103.2813625.

[69] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The long "Taile" of typosquatting domain names. In *23rd USENIX Security Symposium (USENIX Security 14)*, 2014. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/szurdi.

[70] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. Invisible probe: Timing attacks with pcie congestion side-channel. In *2021 IEEE Symposium on Security and Privacy*, 2021. doi:10.1109/SP40001.2021.00059.

[71] The Chromium Project. Chromium. https://github.com/chromium/chromium.

[72] The HTTP Archive. https://httparchive.org/.

[73] WebKit. A fast, open source web browser engine. https://webkit.org/.

[74] Webrecorder. pywb. https://github.com/webrecorder/pywb.

[75] WHATWG. HTML Standard: Link type "preload". https://html.spec.whatwg.org/multipage/links.html#link-type-preload.

[76] WHATWG. HTML Standard: Parsing HTML documents. https://html.spec.whatwg.org/multipage/parsing.html.

[77] World Wide Web Consortium (W3C). WebDriver. https://www.w3.org/TR/webdriver1, 2018.

[78] World Wide Web Consortium (W3C). Mixed Content. https://w3c.github.io/webappsec-mixed-content, 2023.

[79] World Wide Web Consortium (W3C). Content security policy (csp). https://w3c.github.io/webappsec-csp/, 2024.

[80] WPT Contributors. web-platform-tests documentation. https://web-platform-tests.org.

[81] Zihao Zhan, Zhenkai Zhang, Sisheng Liang, Fan Yao, and Xenofon Koutsoukos. Graphics peeping unit: Exploiting em side-channel information of gpus to eavesdrop on your neighbors. In *2022 IEEE Symposium on Security and Privacy*, 2022. doi:10.1109/SP46214.2022.9833773.

[82] Jiang Zhang, Konstantinos Psounis, Muhammad Haroon, and Zubair Shafiq. Harpo: Learning to subvert online behavioral advertising. In *Proceedings of the Network and Distributed System Security Symposium 2021*, 2021. doi:10.14722/ndss.2022.23062.

A De-Obfuscated Google Analytics Code

Google Analytics uses programmatically added event listeners for an HTML node to react to events while still allowing CSP to block inline scripts (see Listing 2).

```
1 ele = document.createElement("script")
2 ele.type = "text/javascript"
3 ele.src = ff.createScriptURL(url)
4 ele.onload = loadHandler
5 ele.onerror = errorHandler
6 ele.setAttribute("nonce", nonce)
7 scr = document.getElementsByTagName("script")[0]
8 scr.parentNode.insertBefore(ele, scr)
```

Listing 2: A deobfuscated example of Google Analytics code programmatically setting event listeners.

B Paper Collection Methodology

We build the paper collection for Section 5 up on the SoK paper by Stafeev and Pellegrino [63], who analyzed the use of crawlers in Web measurement studies. They include USENIX Security, S&P, NDSS, CCS, as well as the specialized venues WWW, PETS, and IMC in their study, and collected all papers from 2010 to 2022 that contain the keywords `tranco`, `alexa` in a combination with `top` or `site`. From the resulting 1,057 papers, they removed all papers that did not employ automated crawling. The remaining papers build a list of 403 Web crawling studies. Taking this list as a base, we analyzed all (137) papers from 2020 to 2022 to understand the individual uses of crawlers and whether or not WebREC could have been used. We also only considered papers that measured data that are within the scope of what WebREC offers, i.e., they either analyze webpage JavaScript executions, the page's content, or any HTTP requests made. This excludes studies using crawlers for other goals, such as network benchmarking purposes [40, 45] or side-channel attacks [70, 81] and narrowed our focus to 97 papers.