To Re-experience the Web: A Framework for the Transformation and Replay of Archived Web Pages

JOHN BERLIN, Old Dominion University, USA
MAT KELLY, Drexel University, USA
MICHAEL L. NELSON, Old Dominion University, USA
MICHELE C. WEIGLE, Old Dominion University, USA

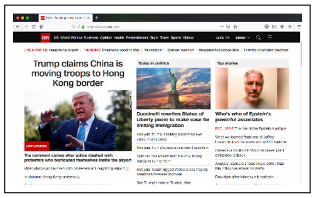
When replaying an archived web page, or *memento*, the fundamental expectation is that the page should be viewable and function exactly as it did at archival time. However, this expectation requires web archives upon replay to modify the page and its embedded resources so that all resources and links reference the archive rather than the original server. Although these modifications necessarily change the state of the representation, it is understood that without them the replay of mementos from the archive would not be possible. The process of replaying mementos and the modifications made to the representations by web archives varies between archives. Because of this, there is no standard terminology for describing the replay and needed modifications. In this paper, we propose terminology for describing the existing styles of replay and the modifications made on the part of web archives to mementos to facilitate replay. Because of issues discovered with server-side only modifications, we propose a general framework for the auto-generation of client-side rewriting libraries. Finally, we evaluate the effectiveness of using a generated client-side rewriting library to augment the existing replay systems of web archives by crawling mementos replayed from the Internet Archive's Wayback Machine with and without the generated client-side rewriter. By using the generated client-side rewriter, we were able to decrease the cumulative number of requests blocked by the content security policy of the Wayback Machine for 577 mementos by 87.5% and increased the cumulative number of requests made by 32.8%. We were also able to replay mementos that were previously not replayable from the Internet Archive. Many of the client-side rewriting ideas described in this work have been implemented into Wombat, a client-side URL rewriting system that is used by the Webrecorder, Pywb, and Wayback Machine playback systems.

1 INTRODUCTION

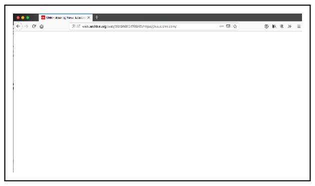
Web archiving is gaining increasing prominence as reliance on the Web for information dissemination and communication has increased. Web pages can be changed or may be removed, and web archives can provide a record of what those web pages originally contained. Journalists and researchers have come to rely on web archives for evidence used to hold public figures accountable [Frew 2022; Internet Archive 2022] and for studying disinformation campaigns [Weigle 2022]. Historians studying the recent past have also come to understand the importance of web archives [Milligan 2019]. It is important, therefore, especially for news and government-related web pages, to have an accurate record of what those pages once contained. Along with capturing the content of web pages, it is also vitally important that the

Authors' addresses: John Berlin, Old Dominion University, Norfolk, VA, USA, n0tan3rd@gmail.com; Mat Kelly, Drexel University, Philadelphia, PA, USA, mkelly@drexel.edu; Michael L. Nelson, Old Dominion University, Norfolk, VA, USA, mln@cs.odu.edu; Michele C. Weigle, Old Dominion University, Norfolk, VA, USA, mweigle@cs.odu.edu.

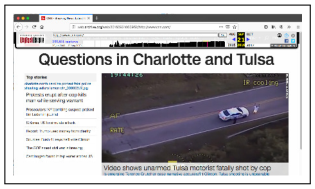
1559-1131/2023/3-ART $15.00
https://doi.org/10.1145/3589206



(a) Live CNN.com, August 13, 2019



(b) August 13, 2019 capture, replayed from the Internet Archive, 2019-08-13


Questions in Charlotte and Tulsa
Video shows unarmed Tulsa motorist fatally shot by cop

(c) September 21, 2016 capture, replayed from the Internet Archive, 2016-09-21

Fig. 1. Screenshots of CNN.com taken August 13, 2019 - (a) live, (b) archived in 2019, and (c) archived in 2016

content in those archived pages is able to be faithfully replayed, allowing for the re-experiencing of the web page at the time of capture.

The Internet Archive (IA),[1] a non-profit digital library, was founded in 1996 with the mission of "Universal Access to All Knowledge", which includes crawling and preserving all public web sites. The goal was not only to simply preserve the Web's content, but to re-experience it through the Wayback Machine,[2] which allows users to replay archived web pages. In addition to IA, many other web archiving initiatives[3] have been created, many of which focus on archiving particular portions of the Web, whether it be public government documents (e.g., the Library of Congress Web Archive[4]) or specific domains (e.g., UK Web Archive[5]). Although the Internet Archive was the first web archive and remains the largest, there are more than a dozen publicly accessible web archives, and many more "dark archives" [Bailey et al. 2013; Farrell et al. 2018; Gomes et al. 2011]. When aggregated, these other web archives can provide coverage of web pages comparable to the Internet Archive [AlSum et al. 2013, 2014].

As web pages have become more complex, both the tasks of archiving and replaying web pages have also become more complex. New strategies must be employed to preserve today's Web [Crook 2009; Masanès 2006; Nelson 2013b, 2020; Rosenthal 2012; Toyoda and Kitsuregawa 2012]. Take, for instance, the issue of "unarchivable" web pages with the seeming disappearance of content from captures of CNN's main web page (*www.cnn.com*) right before the US Presidential Election of 2016 [Berlin 2017]. Figure 1 shows screenshots of a live CNN page from August 13, 2019, the archived version of that page as replayed from the IA's Wayback Machine in 2019, and an archived page from before November 2016 replayed in 2019. The primary issue is with the changes CNN made to their content delivery network (CDN) and how the Wayback Machine replays the page's JavaScript, which retrieves the page's resources from the CDN. The web page from 2019 (and those from November 2016 until Spring 2020) was in fact archived, but the issue was that it was not replayable for an extended period of time. In this paper, we introduce a technique for client-side modifications that allows this archived web page to be replayed.

Not all web archives use the same technology for capture or for replay. Replay can affect how users perceive the quality of an archive in the same manner as capture does. Sometimes, if all of the resources have been captured, broken replay can be fixed. But first, we must describe the ways in which various archives perform replay. The web archiving community currently lacks the terminology to describe the

[1]https://archive.org/
[2]http://web.archive.org/web/20020806031346/http://www.archive.org:80/wayback/press_kit/press_release.html
[3]https://en.wikipedia.org/wiki/List_of_Web_archiving_initiatives
[4]https://www.loc.gov/programs/web-archiving/about-this-program/
[5]https://www.webarchive.org.uk

existing styles of replay and the modifications made to an archived web page and its embedded resources to facilitate replay. This paper aims to address this issue and makes the following contributions:

- Provides a classification of and terminology for the current styles of replay
- Proposes a standard and generalized method for the generation of client-side rewriting libraries
- Details a combination server-side and client-side rewriting technique that decreases the number of modifications made to archived JavaScript and provides an archive with more control over replay
- Evaluates the effectiveness that client-side rewriting would have in augmenting already existing server-side rewriting systems of an archive

This paper is based on Berlin's 2018 Masters thesis [Berlin 2018], which provides full details of this work. In August 2017, both the Webrecorder and Pywb playback systems adopted the client-side rewriting methods described in this work.[6] Pywb's client-side rewriter was then split out into the Wombat client-side URL rewriting system [Kreymer 2019] in 2019. Wombat is an essential part of many web archive replay systems, including the Internet Archive's Wayback Machine. Further, in the Spring of 2020, because of Berlin's work, the Internet Archive updated their Wayback Machine to resolve the CNN replay issue described above.[7]

We have organized this overview of the issues and approaches for client-side rewriting as follows. Section 2 covers background on the Memento protocol and archiving and replay of web pages. Section 3 discusses related research on archiving and replaying dynamic content and related security issues for web archives. Section 4 describes the current methods of web archive replay, including URI rewriting and further modifications necessary for archive replay, and introduces our terminology for the different styles of web page replay. Section 5 presents our approach for client-side re-writing to overcome some of the replay issues discussed. Section 6 presents an evaluation of our approach. Finally, Section 7 offers a conclusion and thoughts on future work.

2 BACKGROUND

2.1 Memento Framework

The Memento Framework [Van de Sompel et al. 2013, 2009] defines inter-archive coordination and provides succinct terminology for referring to archived resources. Each resource on the web is identified by a URI; when referring to URIs using the Memento framework's terminology, it is a URI-R. A **URI-R** "is used to denote the URI of an Original Resource," a **URI-M** "is used to denote the URI of a Memento," and a **memento** represents "an Original Resource as it existed at time T" [Van de Sompel et al. 2013]. The time T is referred to as the **Memento-Datetime**.

A TimeGate (URI-G) is a resource that will select (negotiate) the closest URI-M for a URI-R based on the datetime supplied in the *accept-datetime* HTTP header. A TimeMap (URI-T) for a URI-R contains a listing of the URI-Ms that an archive has in a machine readable format. The TimeGate utilizes these listings to look up the closest datetime of the URI-Ms an archive contains for a particular URI-R. The relationships between mementos, TimeMaps, and the TimeGate are important as they provide a basis for this work, which is focused on replaying (viewing) mementos of an archived web page using a browser.

When an archived web page is replayed in the browser, the browser must dereference the page and any embedded resources within the page. Since the web page is archived, any embedded resources should be loaded from the archive (using URI-Ms) instead of the live web (using URI-Rs). This results in a *composite memento*, which is "a root URI-M and all embedded URI-Ms required to recompose the presentation at

[6]https://github.com/Rhizome-Conifer/conifer/commit/12e2b507b88c4f0c00f29589436f7385dc512f9a (In 2020, Webrecorder was renamed to Conifer.)
[7]https://twitter.com/phonedude_mln/status/1270759972553527297

the clients" [Ainsworth et al. 2014]. In other words, a composite memento encompasses the state and composition of a web page as it potentially existed at preservation time when accessed by the browser.

2.2 Web Page Structure

The Hyper Text Markup Language (HTML) [WHATWG Working Group 2022] is the most popular document type for representing web pages, but HTML only conveys the structure of the web page, relying on a third party to render its content. Each web page contains many HTML elements (tags) in any order that the author of the page desires, but there are three main structural elements. At the root of a HTML document is the HTML tag that can have two children, the `head` and `body` elements. The `head` element can contain elements for defining metadata about the document as well as elements for linking to external resources for the document via the `link` tag and/or embedding JavaScript via the `script` tag. The `body` element contains the primary contents of the HTML document and can contain many HTML elements or text in any order its creator desires. Web page creators can embed images in the document via the image (`img`) tag, video via the `video` tag, or provide navigation to another web page or another section in the document via the anchor (`a`) tag. Other content can be embedded in the document include other documents via the `iframe` tag or additional JavaScript via the `script` tag.

2.3 Archiving and Replay of Web Pages in the Internet Archive

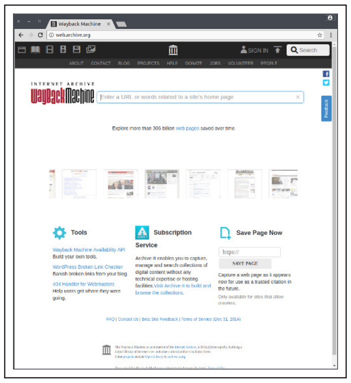
The Internet Archive primarily uses Heritrix for its archival crawler [Mohr et al. 2004], although browser-based crawling is becoming more frequent via their "Save Page Now" interface [Graham 2019]. Heritrix extracts all the URI-Rs contained within a web page [Stern 2011] but without rendering the page or executing the page's JavaScript. Heritrix saves HTTP responses in Web ARChive (WARC) files [ISO 28500 2009], which stores the HTTP requests and corresponding responses during crawling. When indexed by the Wayback Machine or similar tools (e.g., OpenWayback [International Internet Preservation Consortium (IIPC) 2015], PyWb [Kreymer 2013], ReplayWeb.page [Kreymer 2020]), the WARC files allow for replay of composite mementos at the time of their capture. It is the job of the replay software to transform the resources to better emulate the past, such as adding branding and navigational banners and rewriting links so they point back into the web archive and not to the live Web.

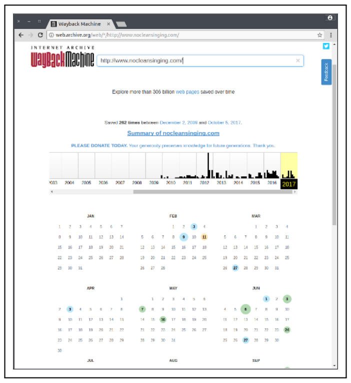
The Internet Archive's Wayback Machine [Fox 2001; Tofel 2007], which is compliant [Internet Archive Developer Portal [n. d.]] with the Memento Framework, is a combination front-end user interface to the contents of the archive and replay engine for viewing its archived web pages. The main page of the Wayback Machine (Figure 2a) allows users to enter a URI-R of a web page to see if it is contained in the archive or select one of the promoted web pages to view. When a user enters a URI-R to view, they are taken to a calendar view (Figure 2b) that displays the date of each memento for the selected year. The user can then select a memento to view (Figure 2c).

2.4 The Broader Web Archiving Ecosystem

The Internet Archive began operation of its web archive in 1996 [Kahle 2021]. For many years, it was the *only* public web archive; the next public web archive, WebCite did not arrive until 2005 and featured the innovation of archiving pages submitted by users [Eysenbach and Trudel 2005]. While WebCite is no longer operational [Aturban et al. 2021], there are many different public web archives in operation [Bailey et al. 2013; Farrell et al. 2018; Gomes et al. 2011]. The International Internet Preservation Consortium (IIPC)[8] is a global coordinating agency where many of the public web archives are represented.

[8]https://netpreserve.org/


(a) Main page http://web.archive.org


(b) Datetime selection http://web.archive.org/web/2017*/http://www.nocleansinging.com/

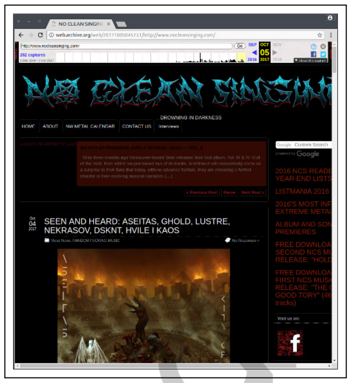

(c) Replay view http://web.archive.org/web/20171005045751/http://www.nocleansinging.com/

Fig. 2. Screenshots of the Internet Archive's Wayback Machine

Web archives have gained societal acceptance [Major and Gomes 2021] and are commonly used to adjudicate matters of law [Eltgrowth 2009; Quarles III and Crudo 2014; Zittrain et al. 2014] and journalism [Boss et al. 2019; Forde et al. 2023; Kriesberg and Acker 2022]. The first scholarly description of web archiving appeared in 2006 [Masanès 2006], and other comprehensive reviews of methods, evaluation, and applications have since appeared [Brügger 2011; Brügger and Milligan 2018; Gomes et al. 2021]. Web archives have long been difficult to use, but on that front a number of computational exploratory environments have been developed to ease that burden, including ArchiveSpark [Holzmann et al. 2016], Archives Unleashed [Ruest et al. 2020], ARCH [Holzmann et al. 2022], and the GLAM-Workbench [Sherratt and Jackson 2020].

The broader acceptance and social impact of web archives, as well as the expanding code base to support their exploration are all welcome developments. However, they do force us to reconsider the fidelity of replayed web pages: if the use case of web archives is no longer limited to novelty of exploring, for example, past GeoCities sites [Lin et al. 2020], and now includes significant legal and financial implications, we must reconsider our approach to how JavaScript is handled on replay. There are simply too many fidelity and security concerns under the current approach; we cannot have sites like `cnn.com` unable to replay for four years.

3 RELATED WORK

This section discusses previous work that has addressed the dynamic nature of the web and/or identified the effects of un-archived resources during replay. There is a trade-off between how web archives should handle JavaScript and the dominant model in most web archives, inherited by the original design of the Internet Archive's Wayback Machine: preserve the JavaScript and at replay time send it back with only its URLs rewritten but otherwise with the same functionality and execution as the JavaScript originally contained on the live Web. This model, described in detail as *Wayback Style* in Section 4, has implications for the performance, completeness, and security of replayed web pages.

3.1 Evaluating Archival Performance

The Archival Acid Test (AAT) [Kelly et al. 2014] is a test suite for evaluating the ability of archival crawlers and replay systems to handle dynamic content. It consists of three categories to test specific aspects of how well an archival crawler can preserve both dynamic and non-dynamic web content. The first category determined a crawler's ability to identify and handle URI variations such as relative (someFile.ext) or scheme-less URIs (//server.com) [Berners-Lee et al. 2005; Nottingham 2014]. The second category examined how well the archival crawler could extract URIs from JavaScript files that were used to bring in additional resources when executed by the browser viewing the test page. The final category tested a crawler's ability to know where to look for URIs used in more complex JavaScript and HTML interactions other than simple DOM manipulation. Most archival crawlers and replay systems tested by Kelly et al. properly handled the first category of tests, but only a few comprehensively handled the majority of the tests. Brunelle et al. [Brunelle et al. 2016] conducted a study of 1,861 URIs with mementos in the Internet Archive between 2005 to 2012 to identify the impact that JavaScript has on the archivability of web pages. The authors found that JavaScript was responsible for 52.7% of all missing resources, and that by 2012, JavaScript was responsible for 33.2% more missing resources than in 2005. Brunelle et al. also observed that JavaScript was used to load 33.7% of all embedded resources in the mementos studied. These findings echoed an earlier study from 2013 [Kelly et al. 2013] that showed that the loss of archived resources increased as JavaScript usage increased over time.

Nearly concurrent with the AAT, Banos & Manolopoulos [Banos and Manolopoulos 2016] created an implementation to evaluate the archivability of live web sites by identifying the facets of the pages, such as their respective accessibility, metadata, and standards compliance. Their approach provided an interactive web site that allows users to enter a URL to evaluate the potential for a URI-R to be archived completely and accurately.

Web archives do not or cannot preserve every resource for every page they archive [Brunelle et al. 2016; Kelly et al. 2013; Leetaru 2015, 2017]. Due to this fact, archived web pages that are missing a portion of their embedded resources appear damaged when replayed. Brunelle, Kelly et al. [Brunelle et al. 2014, 2015] looked at the proportion of missing resources for mementos to assess their damage, finding that the users' perception of damage is a more accurate metric for judging archival quality than the proportion of missing resources. This is an important finding because, depending on how damaged a memento is when replayed, the appearance of missing resources may cause one to believe that the archive has tampered with the memento in some way.

Reyes Ayala [Reyes Ayala 2022] took a grounded theory approach toward evaluating the quality of web archives as they appear in correspondence to their live Web counterparts, relevance of an archived page to its basis, and the qualities of a page that affect its archivability. The work mainly focused on the correspondence aspect with an emphasis on various effects of JavaScript as they are perceived by the user viewing a memento.

The Memento Tracer framework [Klein et al. 2019] extends on some of the work in the process of automating archiving of dynamic web content. While contemporary efforts to preserve dynamic content has improved beyond the crawler capability in the Archival Acid Test [Kelly et al. 2014], the ability to perform high-fidelity captures was either a manual process or did not computationally scale due to the overhead of improving archival quality. Tracer provides both an abstracted method for performing dynamic operations as well as a means of evaluating archival quality.

3.2 Handling The Replay Of Dynamic Content

Current strategies for handling replay of dynamic content rely on client-side intervention either indirectly or directly. The indirect strategy as described by Alam et al. [Alam et al. 2017] involves the usage of a ServiceWorker. The ServiceWorker added by the archive is essentially an added embedded resource for the page capable of intercepting the HTTP requests made by the currently replayed page. This capability of ServiceWorkers allows them to be utilized by the archive to rewrite any URI-R that was either missed by the archive's initial URL rewriting or that was dynamically generated by archived JavaScript to point to the live Web rather than the archive. The direct approach utilizes the client-side archival rewriting JavaScript library Wombat[9]. Wombat is utilized by Pywb and Webrecorder[10] to override the JavaScript APIs of the browser to rewrite any unwritten URI-Rs into URI-Ms. Wombat includes a full URL rewriting system that utilizes overrides of the JavaScript Web and DOM APIs to rewrite any URIs that were missed during the server-side rewriting process. Even though usage of these strategies does not necessarily guarantee a decrease in the proportion of missing resources to non-missing resources of a replayed memento, these strategies seek to increase the viewer's perception of archival quality.

3.3 Security and The Archive

A result of the dominant Wayback Machine replay model is that JavaScript is archived and replayed in the browser, with modifications limited to rewriting URLs so they point back into the web archive and not to the live Web. This preserves JavaScript UI functionality, but comes at the cost of potentially replaying web pages that never existed, as well as introducing security vulnerabilities.

The first documented example of unarchived resources altering the validity of an archived page was the "leakage" of live Web resources into the replay of mementos, termed "zombies in the archive" [Brunelle 2012; Nelson 2014]. The term "zombies" is used to refer to live Web resources that leak into replay, in that mementos are considered "dead", whereas live Web resources are "alive". As discussed in Section 3.1, the primary cause of live leaking "zombie" resources in the archive is JavaScript. URI-Rs dynamically created by JavaScript were not rewritten to point back into the archive, and instead pointed to the live Web. When replaying a memento containing zombies, the zombie resources may give the false appearance that the archive has tampered or altered the memento or its embedded resources.

Combining resources from the live Web and the archived Web can be construed as problem of temporal coherence. Ainsworth et al. [Ainsworth et al. 2014, 2015] examined this problem further, looking at the temporal coherence of composite mementos even when all resources are from the archived Web. Temporal incoherence may make it appear as if the archive had maliciously modified the memento or its embedded resources. Ainsworth et al. defined five states of temporal coherence. The first state, called *Prima Facie Coherent*, represents when an embedded memento exists in the archive as it does on the live Web. The second, called *Prima Facie Violative*, represents when an embedded memento does not exist in the archive as it did on the live Web. The third, called *Possibly Coherent*, represents when an embedded memento might have existed in the archive as it does on the live Web. The fourth, called *Probably Violative*, represents an embedded memento that likely did not exist as archived in comparison to the root memento. The final state, *Coherence Undefined*, is used to denote when there is not enough information to accurately determine the memento's coherence state.

Where early work considered zombies and temporal coherence as simply a problem in crawling in replay, Lerner et al. [Lerner et al. 2017] was one of the first to realize that these were security vulnerabilities that

[9]https://github.com/ikreymer/pywb/blob/master/pywb/static/wombat.js
[10]https://webrecorder.io/

could be turned against the web archive itself by writing JavaScript specifically designed to exploit these vulnerabilities. They described three attacks perpetrated by *users* of the web archive. The first attack, called *Archive-Escapes*, highlighted how the URL rewriting performed by the archive does not necessarily rewrite URLs generated by JavaScript; this could be exploited to bring in zombies from the live Web. The second, called *Same-Origin Escapes*, describes how embedded resources normally disallowed from interacting with the embedding page because they come from another origin than the page can interact with the embedding page because the content is replayed from a single origin (the archive). The final attack, *Never-Archived Resources and Nearest-Neighbor Timestamp Matching*, describes how the archive's inability to archive or rewrite dynamically-added resources could be exploited by identifying an archived page with missing resources that come from a domain that is unowned. Through purchasing said domain, the attacker would cause the archive to replace those missing resources with their own. The solutions posed by Lerner et al., namely archival modification of JavaScript at replay time and the separation of replayed content from the archive's presentational components of replay, parallel the existing replay strategies employed by Webrecorder[11] and Perma.cc [Kreymer 2020].

Roughly simultaneously with Lerner et al., Cushman & Kreymer enumerated similar threats against web archives [Cushman 2017; Cushman and Kreymer 2017; Rosenthal 2017], though their terminology and granularity varied slightly. They also described mitigation strategies for these attacks, some of which has influenced the design and implementation of other public web archives, such as Perma.cc [Zittrain et al. 2014].

Watanabe et al. [Watanabe et al. 2020] generalized the work of Lerner et al. and Cushman & Kreymer to be inclusive of any sites that re-host content, including web archives, language translators (e.g., Google Translate), and privacy proxies (e.g., ProxySite). These re-hosting sites are susceptible to attacks on a client viewing the site in a manner that may not have existed on the basis site. The typical mechanisms of security that prevent JavaScript-driven interaction, like cross-origin resource sharing (CORS) [van Kesteren 2020], are not applicable when pages from different sites are simultaneously hosted on the same re-hosting site. The restructuring of multiple sites (e.g., URI-Rs) being re-hosted at one site (e.g., archive.org) produced the "melting pot" effect. This effect allowed interaction of archived sites that would have previously been hosted at different domains and thus had the same origin policy (SOP) in place.

Breaking from the Wayback Machine model of simply replaying JavaScript, an alternative approach for JavaScript was proposed by Goel et al. [Goel et al. 2022], where they focus on identifying and eliminating JavaScript that will not work correctly (e.g., handing writes back to the server) or will not be executed during replay (e.g., handling a server push that will never come). Their goal is primarily about optimizing the crawling, storage, and replay requirements for archived web pages, but it does represent a break from the approach of most web archives in attempting to modify JavaScript for performant replay.

The JavaScript strategy that is the most divergent from the Wayback Machine is the Archive.today web archive [Nelson 2013a], which strips out all JavaScript from its storage and replay, saving only the resulting DOM and not the original HTML and JavaScript. This approach eliminates all security vulnerabilities associated with replaying JavaScript, but comes at the cost of losing UI functionality often implemented with JavaScript (e.g., navigation, pan, zoom).

4 STYLES OF WEB ARCHIVE REPLAY

Because of the prevalence and longevity of the Internet Archive's Wayback Machine, we first categorize the styles of web archive replay into *Wayback style* and *Non-Wayback style*. In *Wayback style* archiving

[11]Since conducting the study described in this paper, the service at webrecorder.io has been renamed "Conifer" while the efforts toward maintaining the software retain the "Webrecorder" name.

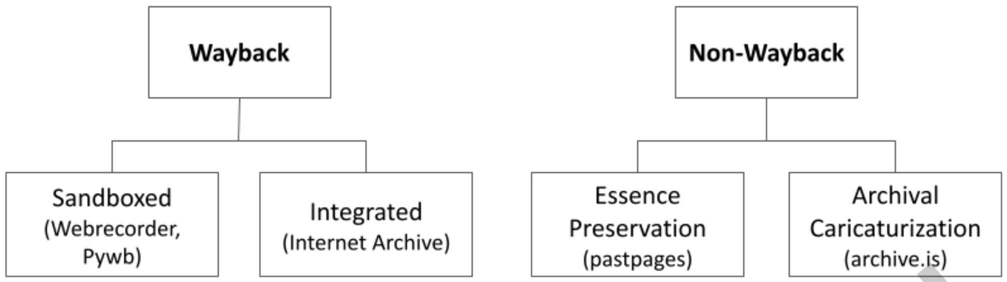

Wayback
Non-Wayback
Sandboxed (Webrecorder, Pywb)
Integrated (Internet Archive)
Essence Preservation (pastpages)
Archival Caricaturization (archive.is)

Fig. 3. Illustrating the various styles of replay

and replay, all of the resources of a web page are captured and stored in WARC files, as described in Section 2.3. Based on the method of replay, the resources in the appropriate WARC files are accessed, modified, and rendered in the viewer's web browser. Figure 3 illustrates our categorization of replay styles, which are detailed in this section.

Archives using the Wayback model must perform certain modifications on the original resource to facilitate proper replay of the archived web page. These modifications may include Archival Linkage modifications (Section 4.1.1) and Replay-Preserving modifications (Section 4.1.2). **Archival Linkage** modifications are made to the URI-Rs found in a memento and its embedded resources so that they no longer link to the live Web but back to the archive. Similarly, archives must perform **Replay-Preserving** modifications to negate intended semantics of specific HTML element and attribute pairs to ensure that replay of the memento is possible.

In addition to modifications of the archived resources, there can also be a difference in how replay systems present the replayed memento in the archive's web page. This can be done through **Sandboxed Replay**, where the replayed archived resources are separated from the archive's user interface using an `iframe`, or through **Integrated Replay**, where the web archive's user interface is inserted into the web page of the replayed memento.

The main Non-Wayback model replay styles are Essence Preservation (Section 4.2.1) and Archival Caricaturization (Section 4.2.2). **Essence Preservation** focuses on capturing only what the web page *looked like* at archival time, and the result of preservation is an image, PDF, or video. **Archival Caricaturization**, an extension of essence preservation, applies a transformation to the page and its embedded resources during preservation such that the archived representation is radically different from the original representation.

4.1 Wayback Style Replay

In this section, we discuss the major modifications applied in Wayback style replay, namely Archival Linkage modifications and Replay-Preserving modifications. We will also discuss the differences between replay systems that employ Sandboxing and those, such as the IA's Wayback Machine, that do not.

4.1.1 Archival Linkage Modifications. To facilitate the replay of mementos, archives must modify (rewrite) the URI-Rs contained in the page and its embedded resources so that they no longer reference (link to) the live Web from where they were archived, but back to the archive. If not for URI rewriting, the embedded resources of an archived web page would continue to reference the live Web, resulting in live Web leakage or "zombies in the archive" (Section 3.3). We define the modifications made by the archive to a page and its embedded resources (including JavaScript and CSS) for replay from the archive as *archival linkage modifications*. This section will describe the methodology used by OpenWayback,[12] the open source implementation of the Wayback Machine, and Pywb.[13] Both OpenWayback and Pywb are derived from the original closed source implementation used by the Internet Archive.

HTML. In HTML, URI-Rs may exist in the markup as the text content or as an attribute value for an element. URI-Rs that are a part of an element's text content do not necessarily need to be rewritten as they are not typically used for any other purpose than to be displayed. Conversely, when used as the value of an element attribute, they are used to provide functionality for the page based on the element or attribute semantics [WHATWG Working Group 2022].

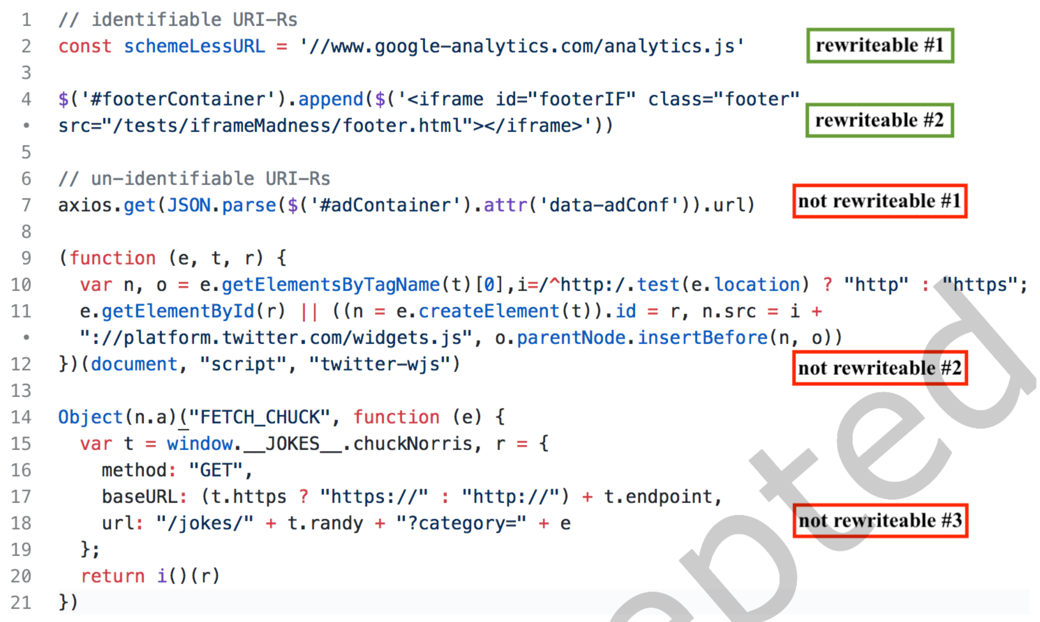
The *href* and *src* attributes are associated with the more commonly used HTML elements. The *src* attribute is used to embed an external resource into the current page based on the tag's semantics. For instance, the `script` tag is associated with embedding JavaScript into a page either by including the JavaScript code as the text contents of the tag or by making the browser fetch the code using the URI-R supplied as the value for its *src* attribute.

The *href* attribute has three purposes based on the semantics of the enclosing tag. The first purpose is to provide navigable links within the document's text through an `a` tag. Rewriting the value of the *href* attribute for these tags allows the viewer of the archived page to stay within the archive when moving between pages rather than going to the live Web. The second purpose for the *href* attribute defines a "base" URI by which all other relative values of the *src* or *href* attribute are relative URIs to be resolved. Rewriting the tag allows the archive to skip rewriting of any relative URI-Rs that come after the tag, because the browser will resolve the un-rewritten relative URIs using the re-written one provided by the `base` tag's *href* attribute. The third and final purpose of the *href* attribute is used by the `link` tag to specify a relationship between the current page and an external resource as defined by an additional attribute. The *rel* attribute of the `link` tag indicates how to interpret the link (*href*) that this tag is making to another resource. Modifications of the `link` tag's *href* are only necessary for *rel* types `stylesheet`, `prefetch`, `preload`, `icon`, `dns-prefetch`, `manifest`, and `import` because they initiate a browser fetch, where the other *rel* types do not [Grigorik 2018].

JavaScript. JavaScript is a dynamic, interpreted programming language that requires execution to determine the ultimate result of the values on which the code operates or produces [Harband et al. 2021]. This requires archives to carefully consider the context in which an URI-R may appear in the JavaScript code along with the *how* and *where* it may be retrieved by the archived JavaScript. Figure 4 shows two examples where it is possible for an archive to rewrite URI-Rs (lines 2 and 4) and three examples of when it is impossible to do so (lines 7, 9-12, and 14-20). The examples that are rewritable do not involve any dynamically computed parts, whereas those that are not rewritable involve dynamically computed values, which are not easily discoverable. The un-rewritable nature of dynamically computed URI-Rs, especially those shown in Figure 4, shows that JavaScript is syntax and interpretation dependent. In line 11, humans can tell by inspection that `n.src` is associated with the JavaScript DOM API for the `script` tag created within the function, but it could have been the *src* attribute of some object not associated with a DOM element.

Cascading Style Sheets (CSS). The final component of archival linkage modifications is modification made to CSS, the language for controlling presentation in HTML documents [W3C 2022]. These changes are trivial in comparison to the linkage modifications made to HTML or JavaScript, as URI-Rs can exist

[12]https://github.com/iipc/openwayback
[13]https://github.com/ikreymer/pywb



```
1  // identifiable URI-Rs
2  const schemeLessURL = '//www.google-analytics.com/analytics.js'
3
4  $('#footerContainer').append($('<iframe id="footerIF" class="footer"
•  src="/tests/iframeMadness/footer.html"></iframe>'))
5
6  // un-identifiable URI-Rs
7  axios.get(JSON.parse($('#adContainer').attr('data-adConf')).url)
8
9  (function (e, t, r) {
10   var n, o = e.getElementsByTagName(t)[0],i=/^http:/.test(e.location) ? "http" : "https";
11   e.getElementById(r) || ((n = e.createElement(t)).id = r, n.src = i +
•    "://platform.twitter.com/widgets.js", o.parentNode.insertBefore(n, o))
12 })(document, "script", "twitter-wjs")
13
14 Object(n.a)("FETCH_CHUCK", function (e) {
15   var t = window.__JOKES__.chuckNorris, r = {
16     method: "GET",
17     baseURL: (t.https ? "https://" : "http://") + t.endpoint,
18     url: "/jokes/" + t.randy + "?category=" + e
19   };
20   return i()(r)
21 })
```

Fig. 4. JavaScript code showing two examples (lines 2, 4) of rewriteable URI-Rs and three examples (lines 7, 9-12, 14-20) that cannot be rewritten without executing the JavaScript code.

in CSS in only two ways (Figure 5): by using the `@import` keyword followed by the name of the style resource to be imported, or by using the `url` keyword. In either case, these URI-Rs are easily identifiable and thus rewritable.

URI Rewriters. To rewrite URIs, archives rely on configurations that list the components of archived resources that might contain URI-Rs that need to be rewritten. The standard attribute rewriter of OpenWayback is used to rewrite the URI-Rs contained in HTML element attributes as URI-Ms. The standard attribute rewriter targets HTML elements that contain *src* and *href* attributes.

Consider the following `link` tag:

```
<link rel="stylesheet" href="/css/style.css">
```

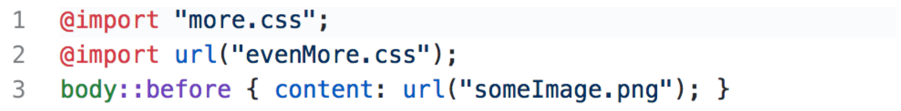
OpenWayback's standard attribute configuration file's entry for the rewriting the `link` tag's *href* for stylesheets is

```
LINK[REL=STYLESHEET].HREF.type=cs
```

Notice that the entry does not explicitly define how the URI-R should be rewritten. Rather, the entry defines a modifier to be added (*cs*) alongside the archival time of the replayed memento. The modifier acts as a hint for the archive to indicate the specific rule-based rewriter to handle the rewriting of the resource. Assuming the datetime of the memento was 20171007035807 and that this should be used in forming the URI-M, the `link` tag would be rewritten as

```
<link rel="stylesheet" href="/20171007035807cs_/css/style.css">
```

Rule-based rewriters, as the name implies, rely on rules to perform the actual rewriting. These rules are derived from the rewrite modifiers added to the URI-Rs by the attribute rewriter and regular expressions. The Pywb stylesheet rewriter defines two rules in the form of regular expressions (`CSS_URL_REGEX` and



```
@import "more.css";
@import url("evenMore.css");
body::before { content: url("someImage.png"); }
```

Fig. 5. URIs in CSS files, specified with the `@import` and url() keywords

`CSS_IMPORT_NO_URL_REGEX`) that cover the different combinations in which URIs can exist in CSS files, examples of which are shown in Figure 5. The regular expressions used by each rewriter must be applied to each line of the file they are rewriting. When one of the rules matches the current line of the file being rewritten, the rule is applied and the rewriter moves on to the next line. If none of the rules match the current line, the rewriter will move on to the next line until one of its rules matches or the rewriter reaches the end of file. This is the totality of the URI rewriting process and is the typical manner that Wayback Machine implementations rewrite the URI-Rs to URI-Ms.[14]

4.1.2 Replay-Preserving Modifications. Replay-preserving modifications are modifications made on the part of an archive to negate the intended semantics of specific HTML element and attribute pairs that would cause the browser to navigate away from the URI-M, apply a security policy not originating from the archive, or to embed a hash of the original representation for resources loaded by the `script` and `link` tags.

The first of these modifications is the negation of `meta` tags that alter the URI-M of the currently replayed page and refreshes the browser, causing navigation to the altered URI-M [Nelson 2014]. For example, the `meta` refresh tag

```
<meta http-equiv="refresh" content="35; url=?refresher=666">
```

defines in the *content* attribute a wait time of 35 seconds before the browser will refresh the page, appending `?refresher=666` to the current URI, thus creating a new URI. When a browser refresh occurs and the URI has changed, this causes navigation to the new URI, which is not desired when viewing the page in the archive. To mitigate the behavior of the `meta` refresh tag, archives can choose to either prefix the *http-equiv* and *content* attributes with an underscore or remove the contents of those attributes.

The second undesired usage of the `meta` tag in replay is to define a Content Security Policy (CSP) without using HTTP headers. Content security policies are used as a defense against malicious content injection (e.g., cross-site scripting) by defining the origins allowed to be loaded on a given page and are typically delivered in HTTP headers of the response [West et al. 2016] for the page to which it should be applied. The Internet Archive, as a direct response to the paper by Lerner et al. [Lerner et al. 2017], now applies their own CSP during replay. CSPs delivered via HTTP are a non-issue for replay, as the original HTTP headers are prefixed by the archive using the convention "X-Archive-Orig" when serving the response on replay [Ainsworth 2015]. The issue arises when `meta` tags are used to define one or more policies for a page. Policies delivered using the `meta` tag are additive to any CSP delivered in the HTTP response for the page. The `meta` tag in Figure 6 defines a policy to restrict resource origins to `http://mydomain.com` only. When archived and replayed, the policy would make the browser refuse to load the embedded resources of both the archive (if any are present) and the archived pages because their replay origin is the archive, not the one listed in the policy. The only possible mitigation is to change the attributes or values of the tag such that the browser's tag and attribute resolution algorithm does not match, or to remove the tag completely.

[14]Non-Wayback Machine archives may do the rewriting differently or not at all (e.g., cached pages in search engines and historical MediaWiki pages).

```
<meta http-equiv="Content-Security-Policy" content="default-src
 ↪   http://mydomain.com; connect-src http://mydomain.com; frame-src
 ↪   http://mydomain.com; img-src http://mydomain.com; media-src
 ↪   http://mydomain.com; object-src http://mydomain.com; script-src
 ↪   http://mydomain.com; style-src http://mydomain.com; font-src data:
 ↪   http://mydomain.com; worker-src http://mydomain.com;">
```

Fig. 6. Content Security Policy defined in a meta tag

```
<link rel="stylesheet" href="theStyleSheet.css" integrity="sha384-eKdwSs2g6PL⌋
 ↪   +F9/RnQ14sov7h5SAFYgq8WJln2tXHOSW/7fJt4G+Td7PcVzkJunk">
<script src="theScript.js"
 ↪   integrity="sha256-qerliYS7q6jrFLa4BJJ3g1ua00vkPz9SjuYsHPLpVzE="></script>
```

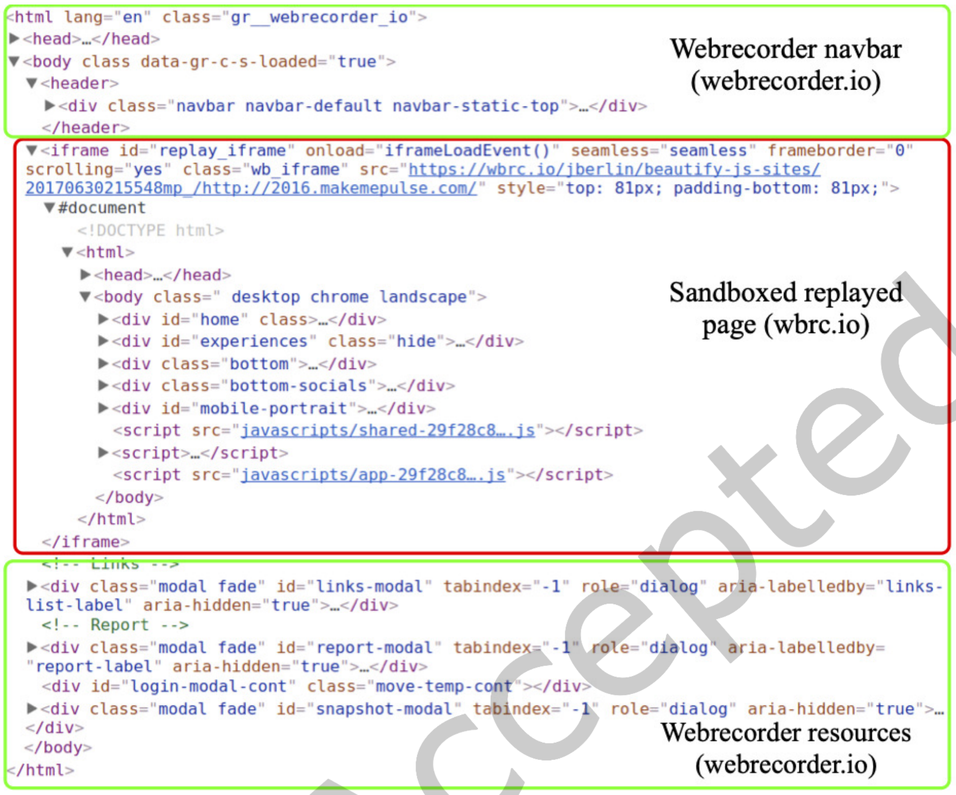
Fig. 7. Integrity attribute of the script and link tags

Similar to the CSP defined in a `meta` tag, which can prevent the browser from loading all embedded resources that do not originate from the specified origin, an integrity attribute can be used to prevent the browser from loading the resources of the `link` and `script` tag. The *integrity* attribute of the `link` and script tags shown in Figure 7 consists of a hash used to validate the hash computed by the browser upon receiving the response body of the resource [Weinberger et al. 2016]. If the browser-computed hash matches the one provided with these tags, then the resource will be loaded otherwise it will not. Due to archives modifying the original resource contents (as described earlier), the hashes will not match and thus, the browser will not load the resource and replay will be affected. Again, the only option to overcome this is to change the attributes such that they do not match the browser attribute resolution algorithm or to remove the attribute.

4.1.3 Sandboxed Replay. As shown in Figure 3, we have further characterized Wayback style replay into **Sandboxed Replay** and **Integrated Replay**. Sandboxed replay separates the replayed page from the archive-controlled portion of the page through isolation. The term *sandboxed* in the case of replay is similar to the architectural design of the Chromium browser [Barth et al. 2008; Reis et al. 2009], which separates the browser kernel (e.g., network and file system stacks) from the rendering engine (e.g., HTML and CSS parsing, image decoding, JavaScript engine). This is a direct reflection of the base assumption that the rendering engine is always compromised [Barth et al. 2008]. Archives and replay systems that employ sandboxed replay share this assumption, i.e., that the pages being replayed are always compromised. There are three main methods by which archives and replay systems typically implement sandboxed replay, though not all use all three methods. *Replay isolation* uses iframes to separate the archive-controlled content from the replayed content. *Temporal jailing* attempts to override certain JavaScript commands to prevent the archived page from navigating to the live web. This requires the use of *client-side rewriting.*

Replay Isolation. Archives and replay systems that employ sandboxed replay, namely Webrecorder and Pywb, do so through **Replay Isolation**. Replay isolation involves the usage of an iframe as the **sandbox** to bring in the actual page being replayed from another domain. This can be seen when considering the memento of `2016.makemepulse.com` on 2017-06-30T21:54:24 when replayed from Webrecorder[15]

[15]https://webrecorder.io/jberlin/beautify-js-sites/20170630215424/http://2016.makemepulse.com/


Webrecorder navbar (webrecorder.io)
Sandboxed replayed page (wbrc.io)
Webrecorder resources (webrecorder.io)

```
<html lang="en" class="gr__webrecorder_io">
▶<head>…</head>
▼<body class data-gr-c-s-loaded="true">
  ▼<header>
    ▶<div class="navbar navbar-default navbar-static-top">…</div>
    </header>
  ▼<iframe id="replay_iframe" onload="iframeLoadEvent()" seamless="seamless" frameborder="0"
  scrolling="yes" class="wb_iframe" src="https://wbrc.io/jberlin/beautify-js-sites/
  20170630215548mp_/http://2016.makemepulse.com/" style="top: 81px; padding-bottom: 81px;">
    ▼#document
       <!DOCTYPE html>
      ▼<html>
        ▶<head>…</head>
        ▼<body class=" desktop chrome landscape">
          ▶<div id="home" class>…</div>
          ▶<div id="experiences" class="hide">…</div>
          ▶<div class="bottom">…</div>
          ▶<div class="bottom-socials">…</div>
          ▶<div id="mobile-portrait">…</div>
           <script src="javascripts/shared-29f28c8….js"></script>
          ▶<script>…</script>
           <script src="javascripts/app-29f28c8….js"></script>
         </body>
       </html>
    </iframe>
    <!-- Links -->
  ▶<div class="modal fade" id="links-modal" tabindex="-1" role="dialog" aria-labelledby="links-
  list-label" aria-hidden="true">…</div>
    <!-- Report -->
  ▶<div class="modal fade" id="report-modal" tabindex="-1" role="dialog" aria-labelledby=
  "report-label" aria-hidden="true">…</div>
    <div id="login-modal-cont" class="move-temp-cont"></div>
  ▶<div class="modal fade" id="snapshot-modal" tabindex="-1" role="dialog" aria-hidden="true">…
  </div>
  </body>
</html>
```

Fig. 8. Example of replay isolation's sandbox on a page replayed by webrecorder.io. The sandboxed portion is outlined in red.

(Figure 8). The two green boxes in Figure 8 represent the non-replay, archive-added portion of replaying web pages, whereas the red box represents the actual replayed page. The top green box highlights the presentation portion of replaying web pages on Webrecorder (i.e., a navbar), whereas the bottom green box highlights additional elements used in conjunction with the navbar. These elements are on the origin `https://webrecorder.io`, and the actual memento being replayed is embedded using an iframe from `https://wbrc.io`. Because iframes bring in what is essentially another browser window into the current one, they have their own security restrictions used by the archive to achieve replay isolation. When the embedded page is from a different origin [WHATWG Working Group 2022] than the embedding page, the content brought in has limited access to the embedding page and vice versa. Much like how the Chromium browser isolates its renderer from the browser kernel, **sandboxed replay** and **replay isolation** ensure the separability of archive-controlled presentational components of replay from the potentially, non-archived controlled replayed page.

Temporal Jailing. Even though an archive may employ **replay isolation**, the archived JavaScript of the replayed page still has the ability to reach out to the live Web [Cushman and Kreymer 2017; Lerner et al. 2017], escaping the sandbox and causing loss of control over replay on the part of the archive. Because of this, an archive using **sandboxed replay** may go a step beyond URI rewriting and **replay isolation** through **temporal jailing**, which is the emulation of the JavaScript environment as it existed at the original Memento-Datetime. This can be accomplished through archive navigational control over the page and client-side rewriting. To achieve archive navigational control, certain JavaScript APIs (namely `document.location`, `window.open`, `document.domain`, `window.location`, `window.history`, and `location`) are overridden by the archive and can no longer navigate the browser away from the archive, open a new window to an unarchived page, or cause a run time error when attempting to change the domain of the page [Berlin 2017]. Rather, when a navigation or history change happens, those events would be reflected in both the replayed page and the archived-controlled portion of sandboxed replay. To accomplish this, any URIs used to cause navigation must be rewritten as URI-Ms, using client-side rewriting.

Client-side Rewriting. Client-side rewriting directly overrides JavaScript web and DOM APIs [WHATWG Working Group 2017a,b, 2022] to provide URI rewrites at the client side. By moving the rewriting client-side, archives ensure that dynamic content added or requested by JavaScript is rewritten. This is essential for providing temporal jailing. We discuss this in more detail in Section 5 by showing how client-side rewriting improves the replay fidelity of mementos replayed from the Internet Archive.

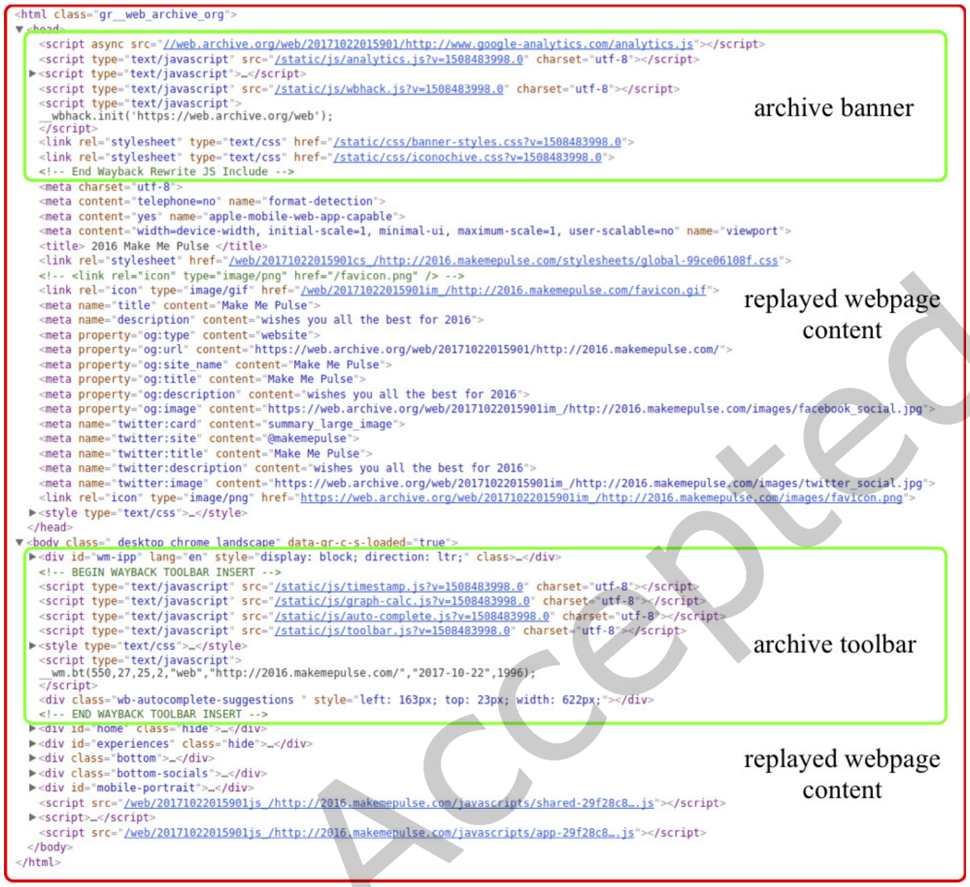
4.1.4 Integrated Replay. Integrated replay is the style of replay that does not separate the replayed memento from the archive-controlled portion of replay. Unlike sandboxed replay, the contents of the replayed memento and the portion added by the archive for replay exist side by side and come from the same domain. Control over the replayed memento is achieved through archival linkage modifications only. To demonstrate this, consider a memento of the same page used to illustrate sandboxed replay, `http://2016.makemepulse.com/` on 2017-10-22T01:59:01Z, when replayed from the Internet Archive. Figure 9 displays how the Internet Archive is inserting its own markup (green boxes) into the HTML belonging to the replayed memento (red box). The markup inserted in the HTML of the replayed memento exists only to provide the banner seen in Figure 10b (green box), which is displayed over the contents of the replayed memento (red box). This can be seen in contrast to the same memento replayed in the sandboxed `webrecorder.io` (Figure 10a), where the archive navigation bar is shown separate from the replayed content. The top frame in integrated replay acts as both the frame containing the archived-added portion of replay and the frame where replay actually occurs. The archived-added portion (green boxes) are not isolated from the replayed memento (red boxes). This leaves the archive-injected markup vulnerable to either direct or indirect tampering by the replayed memento simply because they exist in the same frame (origin), `web.archive.org`.

4.2 Non-Wayback Style Replay

As mentioned earlier, the two main forms of Non-Wayback Style replay are essence preservation and archival caricaturization.

4.2.1 Essence Preservation. Essence Preservation preserves only what the web page looked like at preservation time. This preservation process typically results in an image, PDF, or video [Nelson 2013b] of the web page being created. The news homepage archiving platform, PastPages,[16] demonstrates this (Figure 11). PastPages was created by Ben Welsh, editor of the *Los Angeles Times Data Desk*, to take screenshots of the home pages for 121 news sites once an hour so that the changes they undergo can be studied. The original contents of those homepages (e.g., HTML and embedded resources) are unavailable in this archive. The images saved by this service only prove that the home pages for 121 news sites did in

[16]http://www.pastpages.org the project ended in 2018 and now resides at https://archive.org/details/pastpages.



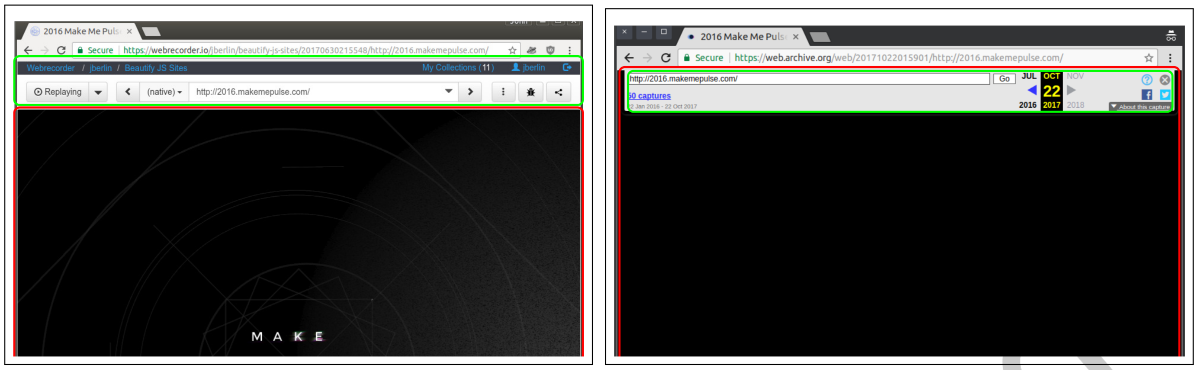
```
<html class="gr__web_archive_org">
▼<head>
   <script async src="//web.archive.org/web/20171022015901/http://www.google-analytics.com/analytics.js"></script>
   <script type="text/javascript" src="/static/js/analytics.js?v=1508483998.0" charset="utf-8"></script>
  ▶<script type="text/javascript">…</script>
   <script type="text/javascript" src="/static/js/wbhack.js?v=1508483998.0" charset="utf-8"></script>
   <script type="text/javascript">
   __wbhack.init('https://web.archive.org/web');
   </script>
   <link rel="stylesheet" type="text/css" href="/static/css/banner-styles.css?v=1508483998.0">
   <link rel="stylesheet" type="text/css" href="/static/css/iconochive.css?v=1508483998.0">
   <!-- End Wayback Rewrite JS Include -->
   <meta charset="utf-8">
   <meta content="telephone=no" name="format-detection">
   <meta content="yes" name="apple-mobile-web-app-capable">
   <meta content="width=device-width, initial-scale=1, minimal-ui, maximum-scale=1, user-scalable=no" name="viewport">
   <title> 2016 Make Me Pulse </title>
   <link rel="stylesheet" href="/web/20171022015901cs_/http://2016.makemepulse.com/stylesheets/global-99ce06108f.css">
   <!-- <link rel="icon" type="image/png" href="/favicon.png" /> -->
   <link rel="icon" type="image/gif" href="/web/20171022015901im_/http://2016.makemepulse.com/favicon.gif">
   <meta name="title" content="Make Me Pulse">
   <meta name="description" content="wishes you all the best for 2016">
   <meta property="og:type" content="website">
   <meta property="og:url" content="https://web.archive.org/web/20171022015901/http://2016.makemepulse.com/">
   <meta property="og:site_name" content="Make Me Pulse">
   <meta property="og:title" content="Make Me Pulse">
   <meta property="og:description" content="wishes you all the best for 2016">
   <meta property="og:image" content="https://web.archive.org/web/20171022015901im_/http://2016.makemepulse.com/images/facebook_social.jpg">
   <meta name="twitter:card" content="summary_large_image">
   <meta name="twitter:site" content="@makemepulse">
   <meta name="twitter:title" content="Make Me Pulse">
   <meta name="twitter:description" content="wishes you all the best for 2016">
   <meta name="twitter:image" content="https://web.archive.org/web/20171022015901im_/http://2016.makemepulse.com/images/twitter_social.jpg">
   <link rel="icon" type="image/png" href="https://web.archive.org/web/20171022015901im_/http://2016.makemepulse.com/images/favicon.png">
  ▶<style type="text/css">…</style>
  </head>
▼<body class=" desktop chrome landscape" data-gr-c-s-loaded="true">
  ▶<div id="wm-ipp" lang="en" style="display: block; direction: ltr;" class>…</div>
   <!-- BEGIN WAYBACK TOOLBAR INSERT -->
   <script type="text/javascript" src="/static/js/timestamp.js?v=1508483998.0" charset="utf-8"></script>
   <script type="text/javascript" src="/static/js/graph-calc.js?v=1508483998.0" charset="utf-8"></script>
   <script type="text/javascript" src="/static/js/auto-complete.js?v=1508483998.0" charset="utf-8"></script>
   <script type="text/javascript" src="/static/js/toolbar.js?v=1508483998.0" charset="utf-8"></script>
  ▶<style type="text/css">…</style>
   <script type="text/javascript">
   __wm.bt(550,27,25,2,"web","http://2016.makemepulse.com/","2017-10-22",1996);
   </script>
   <div class="wb-autocomplete-suggestions " style="left: 163px; top: 23px; width: 622px;"></div>
   <!-- END WAYBACK TOOLBAR INSERT -->
  ▶<div id="home" class="hide">…</div>
  ▶<div id="experiences" class="hide">…</div>
  ▶<div class="bottom">…</div>
  ▶<div class="bottom-socials">…</div>
  ▶<div id="mobile-portrait">…</div>
   <script src="/web/20171022015901js_/http://2016.makemepulse.com/javascripts/shared-29f28c8….js"></script>
  ▶<script>…</script>
   <script src="/web/20171022015901js_/http://2016.makemepulse.com/javascripts/app-29f28c8….js"></script>
  </body>
</html>
```

Fig. 9. Example of integrated replay. The replayed page's contents are outlined in red. https://web.archive.org/web/20171022015901/http://2016.makemepulse.com/

fact exist and that what was captured in the screenshot was displayed to the tool or entity that took the screenshot, i.e., their essence.

4.2.2 Archival Caricaturization. *Archival Caricaturization* is a style of preservation that does not preserve the web page as it *originally was* at some point of time, but rather focuses on only preserving what it *looked like* at some point of time. Caricature is defined as "exaggeration by means of often ludicrous distortion of parts or characteristics"[17] and is the pivotal distinction for this style of preservation. An archive that preserves through caricature is one that applies a derivative transformation to the web page's original markup such that it conforms with the presentational style of the archive but the source is unrecognizable from the original. An archive may choose to keep the original appearance of the web page at the point in time it was preserved or may choose only to preserve certain aspects of the web page like its text, images, and/or video, presenting them in a medium differing from the original. Ultimately,

[17]https://www.merriam-webster.com/dictionary/caricature


2016 Make Me Pulse
https://webrecorder.io/jberlin/beautify-js-sites/20170630215548/http://2016.makemepulse.com/
Webrecorder / jberlin / Beautify JS Sites
My Collections (11)
Replaying
(native)
http://2016.makemepulse.com/
MAKE
https://web.archive.org/web/20171022015901/http://2016.makemepulse.com/
http://2016.makemepulse.com/
Go
JUL OCT NOV
22
2016 2017 2018
50 captures

(a) Sandboxed replay at webrecorder.io

(b) Integrated replay at the Internet Archive

Fig. 10. Examples of the same memento replayed using (a) sandboxed replay at webrecorder.io and (b) integrated replay at the Internet Archive. In both images, the archive portions are outlined in green and the replayed portions are outlined in red.

one cannot retrieve the original web page and its embedded resources at archival time from an archive using this style of preservation.

In order to understand *Archival Caricaturization* replay, consider the example in Figure 12 that demonstrates three different ways to make a `div` have a green background. The first `div` in Figure 12 (line 22) has the green background definition inline, via the *style* attribute of the element. The second `div` (line 23) relies on the CSS id selector "#gbg" with the *pseudo-element* "::before" (lines 11-18) to give it the green background. Note that *pseudo-elements* are considered virtual markup and only become realized for a page after a browser has interpreted a CSS file or `style` tags contents. The third and final `div` (line 24) receives its green background through the usage of the CSS class "greenBG". The visual representation of the example from Figure 12 is shown in Figure 13.

Consider the HTML of the example webpage after being archived in `archive.is`, which uses *Archival Caricaturization* (Figure 14). The example's original markup (Figure 12, lines 1-26) has been replaced completely and is represented by a `div` with the class html1 (Figure 14, lines 14-21). The `head` tag and its content no longer exist, whereas, the `body` tag is represented by another `div` with the class body (lines 16-22). The first green box, line 17, almost exists as it did in the original (Figure 12 lines 22) with the additional style definition "text-align: left;" added to it as do the remaining two green boxes (Figure 12 lines 23 and 24). Although the HTML has been dramatically changed, the rendering of the webpage still appears that same as the original. The archived page can be viewed at http://archive.is/t0T8m.

5 CLIENT-SIDE REWRITING

As discussed in Section 4.1.3, Wayback-style Sandboxed replay systems rely on client-side rewriting to ensure that dynamic content added or requested by JavaScript is rewritten, which is essential for temporal jailing. To illustrate the importance of client-side rewriting, consider the lazy loading of new story footer images on the archived BBC page, http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868, in the Internet Archive as replayed in 2018.[18] The

[18]This issue has since been resolved, likely due to updates based on this work that we provided to the wombat.js client-side rewriting library that the Wayback Machine has since deployed (https://twitter.com/IlyaKreymer/status/1396583618911240193).

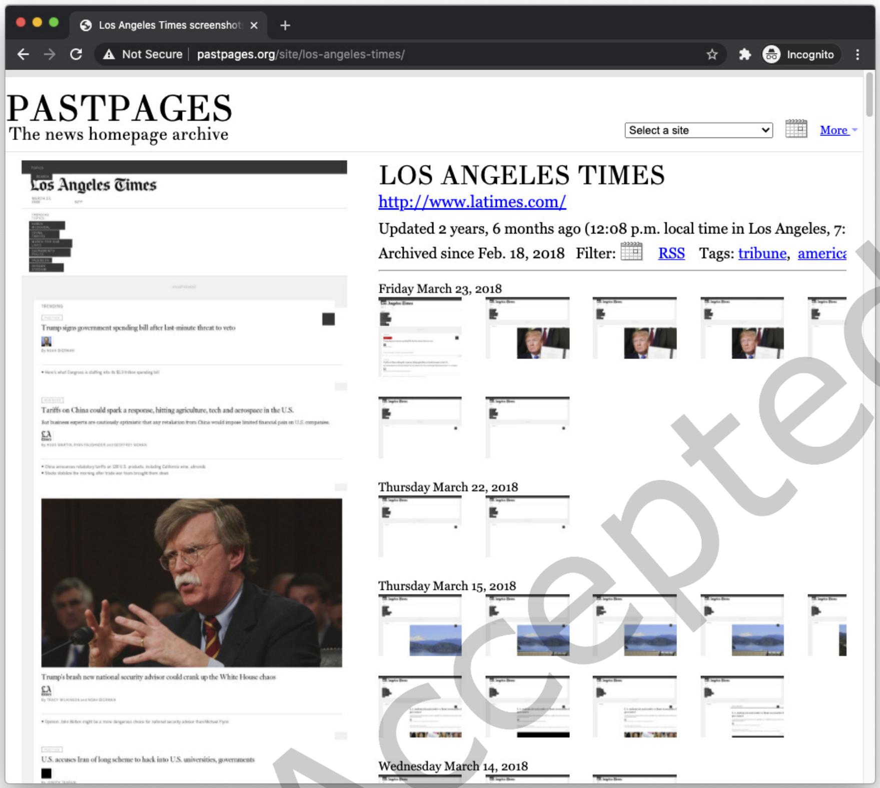

Los Angeles Times screenshot
Not Secure | pastpages.org/site/los-angeles-times/
Incognito
PASTPAGES
The news homepage archive
Select a site
More
LOS ANGELES TIMES
http://www.latimes.com/
Updated 2 years, 6 months ago (12:08 p.m. local time in Los Angeles, 7:
Archived since Feb. 18, 2018 Filter: RSS Tags: tribune, america
Friday March 23, 2018
Thursday March 22, 2018
Thursday March 15, 2018
Wednesday March 14, 2018

Fig. 11. PastPages preserves a news site's essence through images

web page appears to replay as expected (e.g., images accompanying the news story are visible), however, when the user scrolls to bottom of the page, images are missing from the additional stories footer, as shown in Figure 15. Upon further inspection of the missing images, the value for their `img` tag's *src* attributes are un-rewritten URLs (one is highlighted in Figure 16). These un-rewritten URLs are also found as the value of the `img` tag's *datasrc* attributes. Because the *src* attribute of those `img` tags were set to an un-rewritten URL, the images they were supposed to be loading were blocked by the Wayback Machine's CSP (as shown in the Developer window in Figure 17). We inspected the CSS classes of the `img` tags for the blocked images and found that the `img` tags were responsive image placeholders that are to be replaced with the real image by JavaScript. When the page loads, the page's JavaScript makes a request for the lazy loaded images. The response is JSON that contains additional information pertaining to the internals of the page and a string of HTML that is added to document in preparation for the lazy loading of the images (Figure 18). However, the server-side rewriting performed by the Wayback Machine is not able to rewrite the JSON found in the response bodies of archived requests. This causes the JavaScript code responsible for the lazy loading of the images to operate on un-rewritten URLs.

```
<html>
<head>
    <meta name="seo" content="index me!"/>
    <style>
        .greenBG {
            background: green;
            height: 25px;
            width: 25px;
            margin: 5px;
        }
        #gbg::before {
            content: "";
            background: green;
            display: block;
            height: 25px;
            width: 25px;
            margin: 5px;
        }
    </style>
</head>
<body>
<div style="background: green; height: 25px; width: 25px; margin:5px;"></div>
<div id="gbg"></div>
<div class="greenBG"></div>
</body>
</html>
```

Fig. 12. Simple Webpage Showing Different Ways to Display a Div with a Green Background http://cs.odu.edu/~jberlin/originalThreeGreen.html

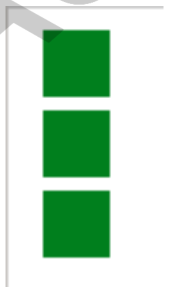
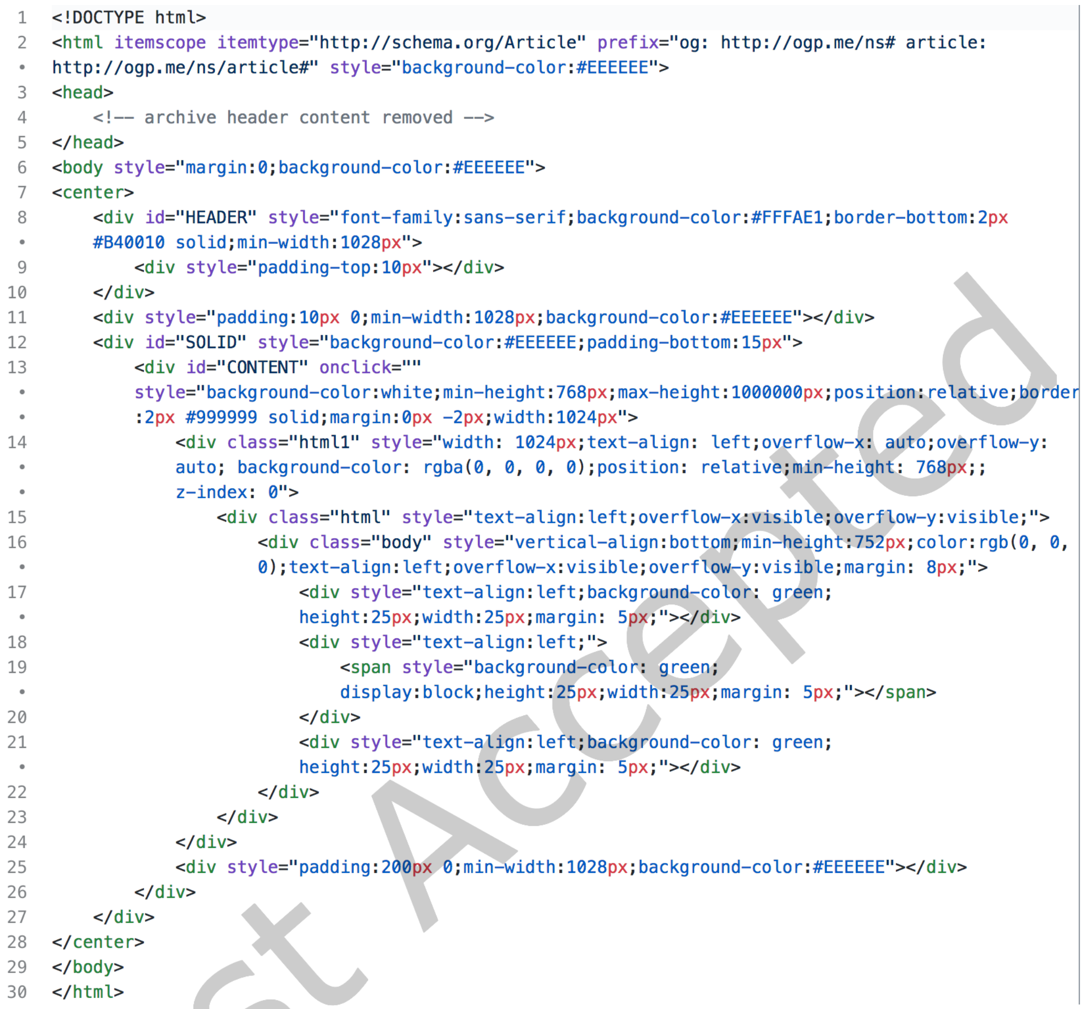


Fig. 13. Webpage from Figure 12 as Rendered in Google Chrome

Client-side rewriting would be able to inspect and rewrite the URL found in the *data-src* attribute in the JSON response, allowing the lazy loading to request the archived images.

The current approach to client-side rewriting is the rewriting library employed by Pywb and Webrecorder Wombat. At its core, Wombat performs the same rewriting done server-side with the addition of targeted JavaScript API overrides in order to rewrite the URLs they operate on. The targeted aspect of Wombat's rewriting makes it an effective addition to server-side rewriting, but it is a handcrafted library specifically tailored for Pywb and Webrecorder, and thus cannot easily be applied to other archival systems. Our goal is to provide a standard general solution for the creation of a client-side rewriting library.



```
<!DOCTYPE html>
<html itemscope itemtype="http://schema.org/Article" prefix="og: http://ogp.me/ns# article:
http://ogp.me/ns/article#" style="background-color:#EEEEEE">
<head>
    <!-- archive header content removed -->
</head>
<body style="margin:0;background-color:#EEEEEE">
<center>
    <div id="HEADER" style="font-family:sans-serif;background-color:#FFFAE1;border-bottom:2px
    #B40010 solid;min-width:1028px">
        <div style="padding-top:10px"></div>
    </div>
    <div style="padding:10px 0;min-width:1028px;background-color:#EEEEEE"></div>
    <div id="SOLID" style="background-color:#EEEEEE;padding-bottom:15px">
        <div id="CONTENT" onclick=""
        style="background-color:white;min-height:768px;max-height:1000000px;position:relative;border
        :2px #999999 solid;margin:0px -2px;width:1024px">
            <div class="html1" style="width: 1024px;text-align: left;overflow-x: auto;overflow-y:
            auto; background-color: rgba(0, 0, 0, 0);position: relative;min-height: 768px;;
            z-index: 0">
                <div class="html" style="text-align:left;overflow-x:visible;overflow-y:visible;">
                    <div class="body" style="vertical-align:bottom;min-height:752px;color:rgb(0, 0,
                    0);text-align:left;overflow-x:visible;overflow-y:visible;margin: 8px;">
                        <div style="text-align:left;background-color: green;
                        height:25px;width:25px;margin: 5px;"></div>
                        <div style="text-align:left;">
                            <span style="background-color: green;
                            display:block;height:25px;width:25px;margin: 5px;"></span>
                        </div>
                        <div style="text-align:left;background-color: green;
                        height:25px;width:25px;margin: 5px;"></div>
                    </div>
                </div>
            </div>
            <div style="padding:200px 0;min-width:1028px;background-color:#EEEEEE"></div>
        </div>
    </div>
</center>
</body>
</html>
```

Fig. 14. Transformation of HTML shown in Figure 12 as archived through caricaturization. http://archive.is/t0T8m

Previously, a general solution for client-side rewriting libraries did not exist because the JavaScript web and DOM APIs provided by the browser could change rapidly without a central standard from which to base the solution, as exists for server-side rewriting. However, we have identified such a standard. The HTML specification [WHATWG Working Group 2022] mandates that even the most reactive web applications and the JavaScript interfaces for interacting with the document (DOM) are governed by the HTML specification, and that user agents supporting web scripting (JavaScript) must abide by the **Web IDL fragments** contained in the specification. Using this knowledge, we can derive a generalized and standard solution for the creation of client-side rewriting.

Taking advantage of Web IDL, we have implemented a client-side rewriter that improves upon that found in the Pywb rewriter. In this section, we present an overview of how our client-side rewriter was implemented. We first describe Web IDL and then the process to auto-generate a client-side rewriter, which

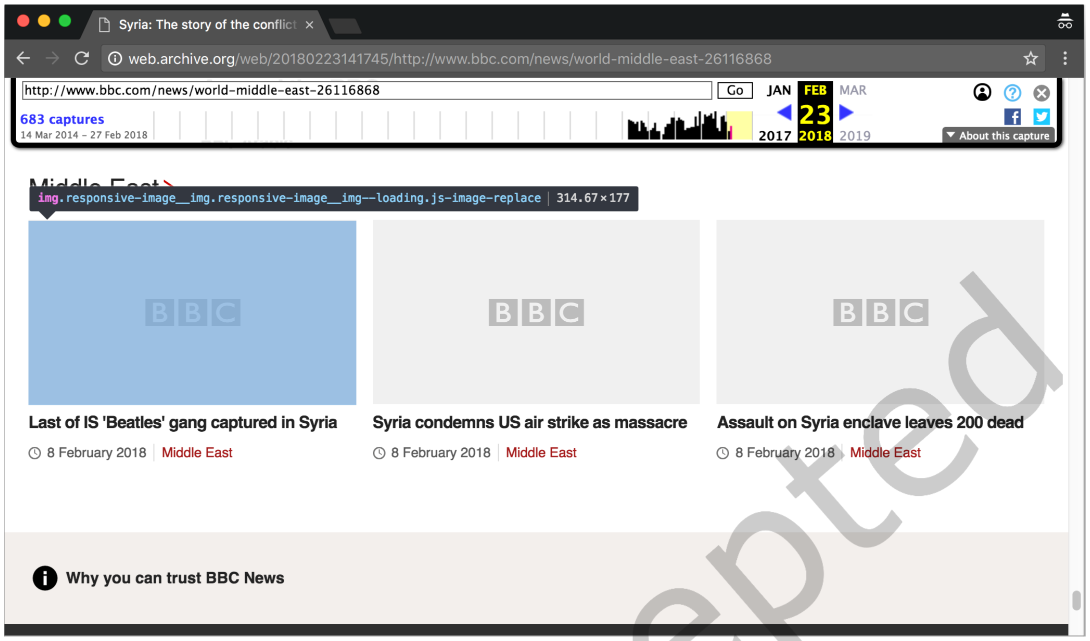
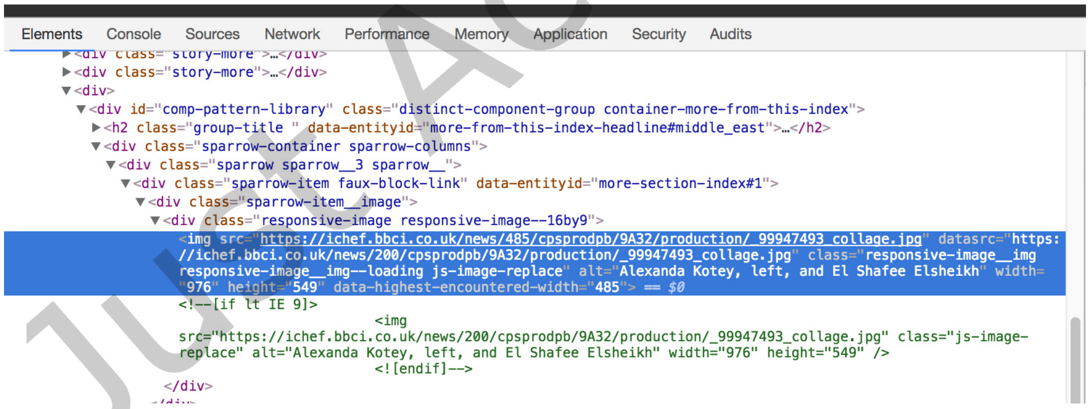

Syria: The story of the conflict
web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868
http://www.bbc.com/news/world-middle-east-26116868
Go
JAN FEB MAR
683 captures
14 Mar 2014 – 27 Feb 2018
23
2017 2018 2019
About this capture
img.responsive-image__img.responsive-image__img--loading.js-image-replace | 314.67×177
Last of IS 'Beatles' gang captured in Syria
8 February 2018 | Middle East
Syria condemns US air strike as massacre
8 February 2018 | Middle East
Assault on Syria enclave leaves 200 dead
8 February 2018 | Middle East
Why you can trust BBC News

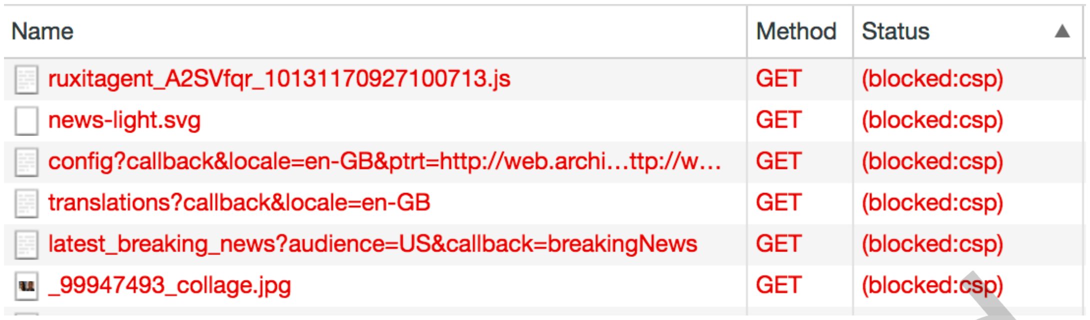
Fig. 15. bbc.com new story footer images blocked by the Wayback Machine's content security policy. http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868



```
    ▶<div class="story-more">…</div>
    ▶<div class="story-more">…</div>
    ▼<div>
      ▼<div id="comp-pattern-library" class="distinct-component-group container-more-from-this-index">
        ▶<h2 class="group-title " data-entityid="more-from-this-index-headline#middle_east">…</h2>
        ▼<div class="sparrow-container sparrow-columns">
          ▼<div class="sparrow sparrow__3 sparrow__">
            ▼<div class="sparrow-item faux-block-link" data-entityid="more-section-index#1">
              ▼<div class="sparrow-item__image">
                ▼<div class="responsive-image responsive-image--16by9">
                    <img src="https://ichef.bbci.co.uk/news/485/cpsprodpb/9A32/production/_99947493_collage.jpg" datasrc="https:
                    //ichef.bbci.co.uk/news/200/cpsprodpb/9A32/production/_99947493_collage.jpg" class="responsive-image__img
                    responsive-image__img--loading js-image-replace" alt="Alexanda Kotey, left, and El Shafee Elsheikh" width=
                    "976" height="549" data-highest-encountered-width="485"> == $0
                    <!--[if lt IE 9]>
                                                     <img
                    src="https://ichef.bbci.co.uk/news/200/cpsprodpb/9A32/production/_99947493_collage.jpg" class="js-image-
                    replace" alt="Alexanda Kotey, left, and El Shafee Elsheikh" width="976" height="549" />
                                                     <![endif]-->
                  </div>
```

Fig. 16. DOM inspector shows unrewritten image URLs

involves identifying the Web IDL interfaces available in the browser and then generating overrides for those interfaces that specify a URL that should be rewritten. Further details are provided in Berlin's MS thesis [Berlin 2018]. Our client-side rewriter, Emu, is available at https://github.com/N0taN3rd/Emu, and an initial implementation of a web crawler using this client-side rewriter is available at https://github.com/N0taN3rd/msThesisCrawler.



Name	Method	Status ▲
ruxitagent_A2SVfqr_10131170927100713.js	GET	(blocked:csp)
news-light.svg	GET	(blocked:csp)
config?callback&locale=en-GB&ptrt=http://web.archi…ttp://w…	GET	(blocked:csp)
translations?callback&locale=en-GB	GET	(blocked:csp)
latest_breaking_news?audience=US&callback=breakingNews	GET	(blocked:csp)
_99947493_collage.jpg	GET	(blocked:csp)

Fig. 17. Network tab of the browser developer tools, showing that one of the bbc.com new story footer images (`_99947493_collage.jpg`) is blocked by the Wayback Machine's content security policy. http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868

```
<div class="sparrow-item__image">
 <div class="responsive-image responsive-image--16by9">
  <div class="js-delayed-image-load"
   data-src="https://ichef.bbci.co.uk/news/200/cpsprodpb/9A32/production/_99947493_collage.jpg"
   data-width="976" data-height="549" data-alt="Alexanda Kotey, left, and El Shafee Elsheikh"></div>
   <!--[if lt IE 9]>
    <img src="https://ichef.bbci.co.uk/news/200/cpsprodpb/9A32/production/_99947493_collage.jpg"
    class="js-image-replace" alt="Alexanda Kotey, left, and El Shafee Elsheikh" width="976"
↪   height="549"/>
     <![endif]-->
    </div>
</div>
```

Fig. 18. Portion of the response to the request for the additional stories images `/news/pattern-library-components?options....` http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868

5.1 Web IDL

The *Web Interface Design Language* (Web IDL) format was created by the W3C to "describe interfaces intended to be implemented in web browsers" [WHATWG Working Group 2017b]. In essence, it provides a standard way to interact with the DOM. Web IDL specifies the underlying behavior (browser implementation) and shape of the JavaScript Web and DOM APIs through five core constructs: `interface`, `typedef`, `enum`, `dictionary`, and `callback`.

Web IDL uses interfaces to describe how the actual JavaScript [Harband et al. 2021] objects implementing the `interface` are to behave, in addition to how to mutate the object's state and invoke the behavior described by the `interface`. The primary members of an `interface` are `attributes`, describing the exposed state of the implementing object, and `operations`, which describe behaviors (methods) that can be invoked on the object [WHATWG Working Group 2017b]. Each of the constructs defined by Web IDL have their own mapping to the JavaScript execution environment. The main construct of concern in this work is the Web IDL `interface` type. The Web IDL specification states that in each JavaScript implementation (web browser) for a set of Web IDL fragments, there will exist a corresponding JavaScript object, and all `interfaces` the implementation supports will be exposed on the global environment object [WHATWG Working Group 2017b].

Figure 19 shows an example of a Web IDL interface to JavaScript mapping. Lines 1 and 2 illustrate an interface having both an `interface object` and an `interface prototype object`. The interface object exists on the global JavaScript environment object (`window`) as a named property, its IDL interface identifier (Figure 19, line 1). The `prototype object` for the `interface` exists as the prototype property on the `interface object` (Figure 19, line 2). When you create a new `HTMLAnchorElement` (Figure 19, line 4) you receive a new instance (a prototype) of the `interface object`. Through this object, you can view or mutate the new instance's state via setting or getting its properties (Figure 19, lines 5 and 6).

```
1  window.HTMLAnchorElement // interface object
2  window.HTMLAnchorElement.prototype // interface prototype object
3
4  let a = document.createElement('a') // new instance
5  a.href = 'http://xyz.com' // mutates the instances state (set)
6  console.log(`href = ${a.href}`) // view the instances state (get)
```

Fig. 19. Web IDL interface (`HTMLAnchorElement`) to JavaScript mapping (a)

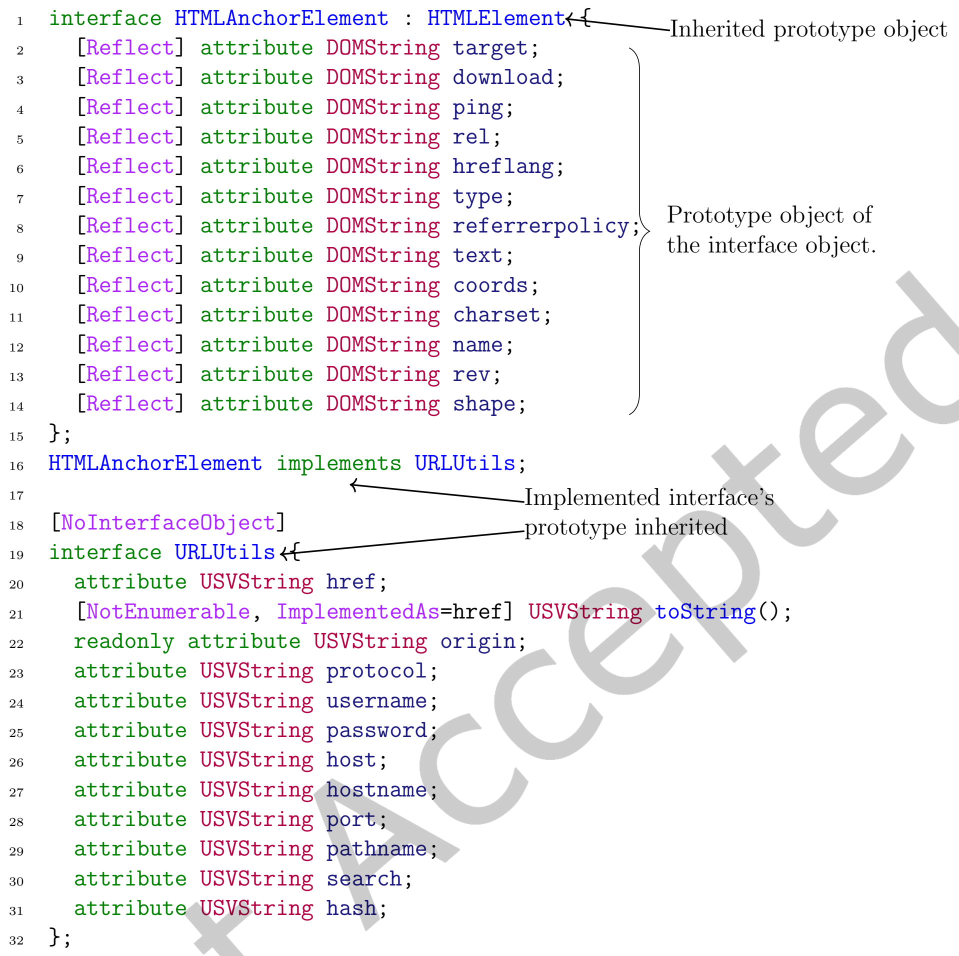
Now consider the actual Web IDL fragment for the `HTMLAnchorElement`, seen with annotations in Figure 20. The definition for the `HTMLAnchorElement` has all the attributes defined for the anchor tag in the HTML specification except for the *href* attribute, found on the `URLUtils` interface which `HTMLAnchorElement` implements (line 16). Because the `URLUtils` interface was defined with the `NoInterfaceObject` extended attribute, it will not have an `interface object`, only an `prototype object`, whereas `HTMLAnchorElement` has both because it was not defined with the `NoInterfaceObject` extended attribute. Because `HTMLAnchorElement` implements `URLUtils`, the attributes and operations on the `URLUtils prototype object` are also on the `HTMLAnchorElement interface object`. Likewise, it also contains the attribute and operations from `HTMLElement`, as it is extended (inherited) by `HTMLAnchorElement`.

5.2 Auto-generating a Client-Side Rewriter

The fundamental goal of URL rewriting is to ensure that every URL found in or operated on by an archived resource points to the archive at a specific Memento-Datetime. Server-side rewriting achieves this goal for archived HTML and CSS because the locations of those URLs are well-defined by their respective specifications. To account for some of the capabilities of JavaScript, server-side rewriting does perform some additional rewriting beyond those well-known locations, but, as shown earlier, there are instances where server-side rewriting is not sufficient. Our implementation of a client-side rewriter takes advantage of the fact that JavaScript can only perform manipulations on the DOM using the APIs described by the Web IDL fragments included or linked to by the specifications for well-known URL identifiers used in server-side rewriting of HTML and CSS. We use these Web IDL fragments in combination with the description of how Web IDL maps to the JavaScript environment to auto-generate a client-side rewriting library.

Automatic generation of a client-side rewriter requires that the rewriter generator have the following properties:

(1) generates JavaScript that follows the convention set by the *de facto* standard client-side rewriting libraries implemented in Pywb and Webrecorder's Wombat



```
interface HTMLAnchorElement : HTMLElement {
  [Reflect] attribute DOMString target;
  [Reflect] attribute DOMString download;
  [Reflect] attribute DOMString ping;
  [Reflect] attribute DOMString rel;
  [Reflect] attribute DOMString hreflang;
  [Reflect] attribute DOMString type;
  [Reflect] attribute DOMString referrerpolicy;
  [Reflect] attribute DOMString text;
  [Reflect] attribute DOMString coords;
  [Reflect] attribute DOMString charset;
  [Reflect] attribute DOMString name;
  [Reflect] attribute DOMString rev;
  [Reflect] attribute DOMString shape;
};
HTMLAnchorElement implements URLUtils;

[NoInterfaceObject]
interface URLUtils {
  attribute USVString href;
  [NotEnumerable, ImplementedAs=href] USVString toString();
  readonly attribute USVString origin;
  attribute USVString protocol;
  attribute USVString username;
  attribute USVString password;
  attribute USVString host;
  attribute USVString hostname;
  attribute USVString port;
  attribute USVString pathname;
  attribute USVString search;
  attribute USVString hash;
};
```

Inherited prototype object
Prototype object of the interface object.
Implemented interface's prototype inherited

Fig. 20. HTMLAnchorElement.idl

(2) knows how to convert the URL identifiers for Web IDL interfaces that are HTML elements (Table 1) and the attributes of the `CSSStyleDeclaration` interface to their server-side equivalents (Section 5.2.1)
(3) knows how to generate the appropriate rewriting functionality client-side for each of the identified interfaces (Sections 5.2.1 and 5.2.2)
(4) understands the inheritance hierarchy and exposed location information included with each interface (Sections 5.2.1 and 5.2.2)
(5) knows how to generate the override modifications in JavaScript (Section 5.2.3)

We will briefly describe how we implemented our rewriter generator to have each of these properties in the following subsections. Additional details are provided in Berlin's MS thesis [Berlin 2018].

Table 1. HTML Element Attributes With Server-Side Rewrite Modifier From Pywb

Tag	Attribute	Rewrite Modifier
a, area	href	None
audio, embed input, source track, video	src	oe_
audio, video	poster	im_
iframe	src	if_
frame	src	fr_
base	href	mp_
form	action	mp_
img	src srcset	im_ im_
link	href	cs_, mp_, None
meta	content	mp_
object	data	oe_
script	src	js_
source	srcset	oe_
*	style	mp_, im_

5.2.1 Identifying Web IDL Interfaces. Fundamentally both client-side and server-side rewriting operate under the same constraints, that is, neither can rewrite URLs unless they know where to look. Server-side rewriting looks for URLs in HTML based off the tag and attribute name (Table 1) and looks for URLs in CSS contained in the import or `url` keywords (Figure 5). Client-side rewriting, on the other hand, must be able to apply overrides for well-known URL identifiers that are found within the constructs of Web IDL. The base set of identifiers required to correctly determine which Web IDL interfaces are needed to generate a complete client-side rewriting library come from the existing URL identifiers used by server-side HTML and CSS rewriting. These 8 URL identifiers are listed in the Attribute column of Table 1. However, the naming conventions for HTML element interfaces in Web IDL do not match the actual tag name, so the Web IDL interfaces must be identified based on if they contain the known attributes.

The Web IDL interfaces associated with these attributes are identified by examining the interface specifications. For example, the *anchor* tag in HTML is denoted by `a`, but the Web IDL interface is `HTMLAnchorElement` (Figure 19). The match is found when considering the identifiers of attributes that are the same as the attribute names used by server-side rewriting (Attribute column in Table 1). The `HTMLAnchorElement` Web IDL interface (Figure 20) includes the attribute *href* through the inherited `URLUtils` interface. Since `HTMLAnchorElement` inherits the prototype object `HTMLElement`, we can add HTMLElement to our list of known Web IDL interfaces associated with the *href* identifier. We use a similar process to discover additional Web IDL HTML element interfaces.

We note that the *style* attribute is also the semantic name of an HTML element whose text content contains the full set of allowed CSS style definitions (example in Figure 21). Because the *style* attribute also doubles as a tag, we know that the generated client-side rewriter must be able to rewrite URLs found in the style definitions of the *style* attribute found on arbitrary elements *and* within the text contents of a `style` tag when modified via JavaScript.

```
<div style="background: url('it.png');"></div>
<p style='cursor: url("cur.svg"), auto;'></p>
<span style='border-image: url("pattern.svg") 40 40 repeat;'></span>
<ul style='list-style: square url("redball.png");'></ul>
<li style='list-style-image: url("liImage.png");'></li>
<style>
    @import "more.css";
    @import url("evenMore.css");
    body::before { content: url("someImage.png"); }
</style>
```

Fig. 21. CSS style properties that may contain URLs and how URLs may exist in CSS style definitions found in a style tag

Table 2. Baseline Interface Discovery Identifiers

Interface kind	Identifiers
HTMLElement	action, content, data, href, poster, src, srcset
Non-HTMLElement	href, url, scriptURL

To identify additional well-known URL identifiers, we consider pages that access and mutate the *style* attribute or any attribute of HTML elements that can be used to initiate browser resource fetches. One example is the `Element` interface which exposes the identifiers `innerHTML`, `outerHTML`, and `insertAdjacentHTML`, used to insert document markup via strings, and the `insertAdjacentElement` identifier used to insert new element instances into the document. Likewise, the `Node` interface exposes identifiers `insertBefore`, `appendChild`, and `replaceChild`, which can also mutate document markup.

The final set of well-known URL identifiers can be found in the naming conventions of the exposed identifiers for non-element interfaces and how the identifiers of the interfaces can be used as the typing of an arbitrary identifier. The identifiers of non-element interface identifiers can be discovered using the naming conventions of their attributes and operation or constructor argument identifiers, *href*, *url*, and *scriptURL*. The downside to using only the identifier names previously mentioned is we would miss the operation identifier `fetch` exposed on the `Window` interface, because neither the identifier itself nor its argument identifiers match any of the well-known identifiers for non-element interfaces. But we were able to overcome this by noticing that the first argument to `fetch` is of type `RequestInfo`, which is a typedef for the `Request` interface or a string. Because we had previously discovered the `Request` interface by its *url* attribute identifier and know the interface is a part of the `RequestInfo` typedef, we can use the typedef and the `Request` interface's type to discover additional identifiers. The type system of Web IDL not only provides a useful heuristic for determining how a discovered identifier is used when name matching is impossible, but also for discovering usages of the type for already discovered interfaces.

This identification process results in a baseline of 10 URL identifiers, 7 for discovering interfaces that are HTMLElements and three identifiers for the discovery of non-HTMLElement interfaces (Table 2). The remaining well-known interface and member URL identifiers shown in Table 3 are considered to be special cases given the specificity of their use cases.

Table 3. Special Well-Known Interface and Member Identifiers

Interface Identifiers	**Member Type**	**Identifiers**
HTMLStyleElement	Attribute	textContent
HTMLIframeElement	Attribute	srcdoc
Attr	Attribute	value, nodeValue
Node	Operation	insertBefore, appendChild replaceChild
Location	Attribute	href
	Operation	assign, replace
Document	Attribute	domain, cookie
	Operation	write, writeln
CSSStyleDeclaration	Attribute	cssText
	Operation	setProperty
Element	Attribute	innerHTML, outerHTML
	Operation	getAttribute, setAttribute insertAdjacentElement insertAdjacentHTML

5.2.2 Automatic Web IDL Interface Identification. Once we have identified the well-known URL identifiers and determined how they map to Web IDL interfaces, we can write an algorithm to automatically identify the Web IDL interfaces that should be evaluated for client-side rewriting.

Automatically identifying relevant Web IDL interfaces can be expressed in two phases: *fragment extraction* and *identification*. In the *fragment extraction* phase (Algorithm 1), each fragment's members from the set of Web IDL fragments to be considered are extracted and the following steps are performed. For each member, we accumulate a mapping of interface identifiers to interfaces, ensuring that partial interfaces are merged into the primary interface and typedefs to the type(s) that were redefined. Then, for each of the extracted interfaces, we merge implemented interfaces into the implementer interface and ensure interfaces inheriting from another are updated to include information about the inheritance hierarchy of which they are a part. Finally, this returns the interfaces and typedefs extracted to the next phase identification. The *identification* phase (Algorithm 2) is multi-part algorithm performing the steps for identifier discovery. For each interface that has an interface object and is an HTMLElement, we check to see if the interface has members matching the well-known HTML identifiers (e.g., *data*, *href*, *src* in Table 2). If the interface is not an HTMLElement, then we check the interface for the non-element identifiers (e.g., *href*, *url*, *scriptURL* in Table 2). This process is outlined in Algorithm 3. A further check is made to determine if the interface is a special case (identified in Table 3) and if so, its members are checked using the associated attribute or operation identifiers (Algorithm 3). Once every interface having an interface object has been checked, we determine which of the identified interfaces are used in a typedef and then check the arguments of operations and constructor of each identified interface to determine if their typing is the typedef. We then determine if any of the identified interfaces have attributes whose type is an already identified interface to ensure we can handle cases such as the location attribute of `Window` and `Document` (Algorithm 4).

Algorithm 1 Web IDL Fragment extraction

```
 1: function GetIdlData(idlFragments)
 2:     interfaces ← Map
 3:     implements, typeDefs,isTypeDefd ← MultiMap
 4:     for each member ∈ extractFragmentMembers(idlFragments) do
 5:         if member ↦ $Interface \vee PartialInterface$ then
 6:             if interfaces.hasKey(member.identifier) then
 7:                 interfaces[member.identifier] ← update(interfaces[member.identifier],member)
 8:             else
 9:                 interfaces[member.identifier] ← member
10:         else if member ↦ $Implements$ then
11:             implements[member.target] ← member.implements
12:         else if member ↦ $TypeDef$ then
13:             typeDefs[member.identifier] ← member.types
14:             for each type ∈ member.types do
15:                 isTypeDefd[type] ← member.identifier
16:     for each identifier ∈ interfaces.keys do
17:         interface ← interfaces[identifier]
18:         if interfaces[identifier].inheritance ! = Nil then
19:             interfaces[identifier] ← updateInheritanceInformation(interface,interfaces)
20:         if implements.hasKey(identifier) then
21:             interfaces[identifier] ← updateFromImplements(interface,interfaces,implements)
22:     return interfaces,typeDefs,isTypeDefd
```

Algorithm 2 Identify interfaces

```
 1: function IdentifyInterfaces(idlFragments)
 2:     interfaces, typeDefs,isTypeDefd ← GetIdlData(idlFragments)
 3:     foundInterfaces ← Map
 4:     for each interface ∈ interfaces do
 5:         if !interface.noInterfaceObject then
 6:             if isOrSubTypeOfHTMLElement(interface) then
 7:                 CheckMemberIdentifiers(interface,htmlElementIds,foundInterfaces)
 8:             else
 9:                 CheckMemberIdentifiers(interface,nonHTMLElementIds,foundInterfaces)
10:             if checkSpecial.hasKey(interface.identifier) then
11:                 SpecialCheck(interface,specialCheck,foundInterfaces)
12:     for each foundId ∈ foundInterfaces.keys do
13:         if isTypeDefd.hasKey(foundId) then
14:             FindTypedefArguments(isTypeDefd[foundId],interfaces,foundInterfaces)
15:     for each foundId ∈ foundInterfaces.keys do
16:         if nonElementInterface(foundInterfaces[foundId]) then
17:             CheckFoundAttsRefFoundType(foundId,interfaces,foundInterfaces)
18:     return foundInterfaces,typeDefs,isTypeDefd
```

These identification algorithms were run on an input set of Web IDL fragments retrieved from the source code repository of the Chromium browser[19] (v67) using the W3C provided Node.js parser.[20] The algorithms successfully identified the HTML element interfaces in which we are interested and the

[19]https://chromium.googlesource.com/chromium/blink/+/master/Source
[20]https://github.com/w3c/webidl2.js

Algorithm 3 Check Interface Based On Identifier Names

```
function CheckMemberIdentifiers(interface,toFind,found)
    attributes ← getAttributesNamed(interface,toFind)
    operations ← getOperationsWithArgsNamed(interface,toFind)
    constructors ← getConstructorsWithArgsNamed(interface,toFind)
    if anyNotEmpty(attributes,operations,constructors) then
        identified ← newIdentified(interface,attributes,operations,constructors)
        found[identified.identifier] ← identified
function SpecialCheck(interface,specialCheck,found)
    attributes ← getAttributesNamed(interface,specialCheck.attributes)
    operations ← getOperationsWithArgsNamed(interface,specialCheck.operations)
    if anyNotEmpty(attributes,operations,constructors) then
        if found.hasKey(interface.identifer) then
            foundIface ← found[interface.identifier]
            foundIface.attributes ← foundIface.constructors ∪ attributes
            foundIface.operations ← foundIface.operations ∪ operations
        else
            foundIface ← newIdentified(interface,attributes,operations)
            found[identified.identifier] ← identified
```

Algorithm 4 Find Interfaces with members typed

```
function FindTypedefArugments(typeDefs,interfaces,found)
    for each typedef ∈ typeDefs do
        for each interface ∈ interfaces do
            operations ← getOperationsWithArgsTyped(interface,typedef)
            constructors ← getConstructorsWithArgsTyped(interface,typedef)
            if operations ! = ∅ ∨ constructors ! = ∅ then
                if found.hasKey(interface.identifier) then
                    foundIface ← found[interface.identifier]
                    foundIface.operations ← foundIface.operations ∪ operations
                    foundIface.constructors ← foundIface.constructors ∪ constructors
                else
                    foundIface ← newIdentified(interface,∅,operations,constructors)
                    found[foundIface.identifier] ← foundIface
function CheckFoundAttsRefFoundType(foundId,interfaces,found)
    for each fId ∈ foundInterfaces.keys do
        atts ← getAttriubteOfType(interfaces[fId],foundId)
        if notEmpty(atts) then
            foundInterfaces[fId].exposesFound[foundId] ← atts
            exposed ← foundInterfaces[foundId]
            exposed.exposedOnFound ← exposed.exposedOnFound ∪ fId
```

named constructor `Audio`, associated with the `HTMLAudioElement`. Table 4 lists the identified Web IDL interfaces and their members.

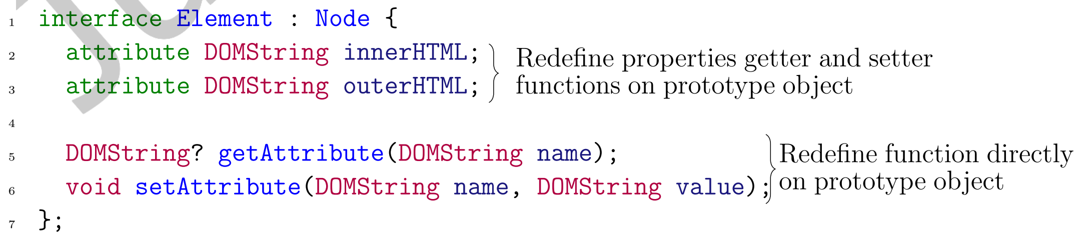
The remaining interfaces and attribute or operation identifiers were discovered using the well-known non-element identifiers in Table 2 and the type matching heuristic (Table 5). The identification algorithm was successfully able to identify the interfaces for using the HTTP protocol namely `fetch`, `Request`, `Response`, `XMLHTTPRequest` and `EventSource` [WHATWG Working Group 2022], the WebSocket protocol [Fette and Melnikov 2011], as well as the location attribute of both `Window` and `Document`.

Table 4. Identified HTML Interfaces from the Chromium browser

Interface	Member
HTMLBaseElement, HTMLAnchorElement HTMLAreaElement, HTMLLinkElement	href
HTMLAudioElement, HTMLEmbedElement HTMLFrameElement, HTMLTrackElement HTMLInputElement, HTMLMediaElement HTMLScriptElement	src
HTMLImageElement HTMLSourceElement	src, srcset
HTMLIFrameElement	src, srcdoc
HTMLFormElement	action
HTMLMetaElement	content
HTMLObjectElement	data
HTMLStyleElement	textContent
HTMLVideoElement	poster, src
HTMLAudioElement	Audio(src)

5.2.3 Generating a Client-Side Rewriter. Generation of the client-side rewriting library relies on the Web IDL to JavaScript mapping discussed in Section 5.1 as the basis for generating the overrides applied to the identified interfaces. Interfaces that are not declared with the `NoInterfaceObject` extended attribute have a corresponding JavaScript object or function object and their attributes and operations exist as named properties on the interface's `prototype object` [Harband et al. 2021; WHATWG Working Group 2017b]. The `prototype object` in JavaScript is the fundamental building block for all objects, and provides a permanent record of an object's own (non-inherited) and inherited properties. Since we know that every identified interface will have a `prototype object`, the modification made to those interfaces can be categorized by five types of overrides: *patch*, *replace*, *replace plus patch*, *foreign substitution*, and *extend*.

The *patch* override patches the `prototype object` of an identified interface that does not expose a constructor. This override redefines the named properties of the interface's attributes and operations in order to intercept un-rewritten URLs (Figure 22). Because the `prototype object` provides a permanent



```
interface Element : Node {
  attribute DOMString innerHTML;
  attribute DOMString outerHTML;

  DOMString? getAttribute(DOMString name);
  void setAttribute(DOMString name, DOMString value);
};
```

Redefine properties getter and setter functions on prototype object
Redefine function directly on prototype object

Fig. 22. Interface attribute and operation *patch* overrides

record for all properties an object has, redefinition of attributes only replaces the original named property's

Table 5. Identified Non-Element or Special Case Interfaces

Interface	Type	Member
Clients	Operations	openWindow
Document	Attributes	location
EventSource	Attributes	url
	Constructors	Constructor(url, eventSourceInitDict)
History	Operations	pushState, replaceState
Navigator	Operations	sendBeacon, registerProtocolHandler isProtocolHandlerRegistered unregisterProtocolHandler
Request	Attributes	url
	Constructors	Constructor(input, requestInitDict)
Response	Attributes	url
	Operations	redirect
ServiceWorkerContainer	Operations	register
ServiceWorkerGlobalScope	Operations	fetch
SharedWorker	Constructors	Constructor(scriptURL, name)
URL	Attributes	href
	Operations	revokeObjectURL
	Constructors	Constructor(url, base)
WebSocket	Attributes	url
	Constructors	Constructor(url, protocols)
Window	Attributes	Location
	Operations	open, fetch
WindowClient	Attributes	url
	Operations	navigate
Worker	Constructors	Constructor(scriptURL)
WorkerGlobalScope	Operations	fetch
XMLHttpRequest	Operations	open

getter and setter functions to use archive-controlled versions. This is the same for the redefinition of named properties for operations, which only replaces the original function's definition with an archive-controlled version (Figure 23).

The *replace* override is used to replace (shadow) the definition of an attribute or operation directly on the existing instance of the interface `Window`, which is the primary global execution object, representing the current browsing context. With the in-place security constraints of the browser, we cannot directly modify the existing `Window` object or its prototype. Rather, we apply the override to the `WindowProxy` object [van Kesteren 2016; WHATWG Working Group 2022], which proxies the exposed attributes and operations of the `Window` interface (Figure 24).

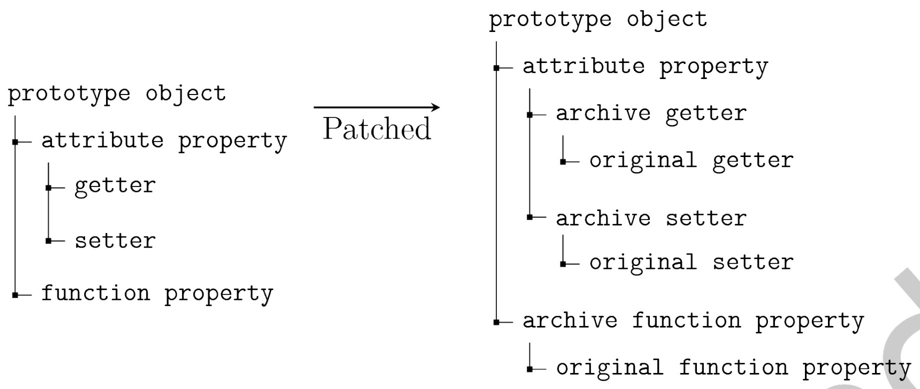


Fig. 23. Prototype object *patch* modification

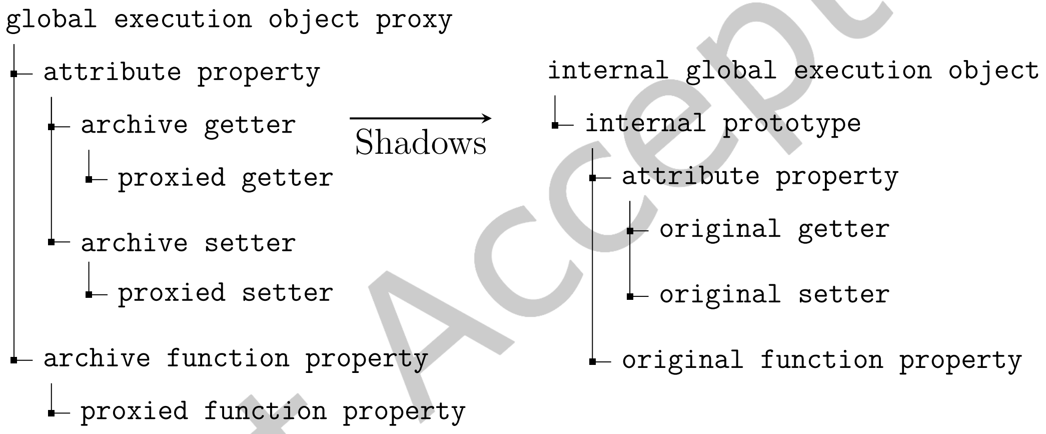


Fig. 24. Global execution object *replace* modification

The *replace plus patch* override is a combination of both the *replace* and *patch* overrides applied to the remaining identified interfaces that have existing instances. For example, the `Document` interface provides two operations for introducing new markup into the current page, namely `write` and `writeln`. We want to ensure that both the existing instance and its prototype object share the same overrides that were made to the existing instance (Figure 25). By replacing the existing instance's copy of the operations and patching the prototype object for the interface, we can ensure that no reference to an unpatched version of the named property exposed by the interface can be retrieved.

The *extend* override creates a new subtype of non-element interfaces that have a constructor or an `HTMLElement` that has a named constructor and replaces the reference to the interface on the primary global object with the archived controlled subtype. The new interface inherits all the properties of the extended interface and is a subtype of the extended interface (Figure 26).

The final override type is *foreign substitution*. It exists primarily due to the capabilities of the unforgeable (unmodifiable or overridable) `Location` interface, which is also an `Unforgeable` attribute of the primary

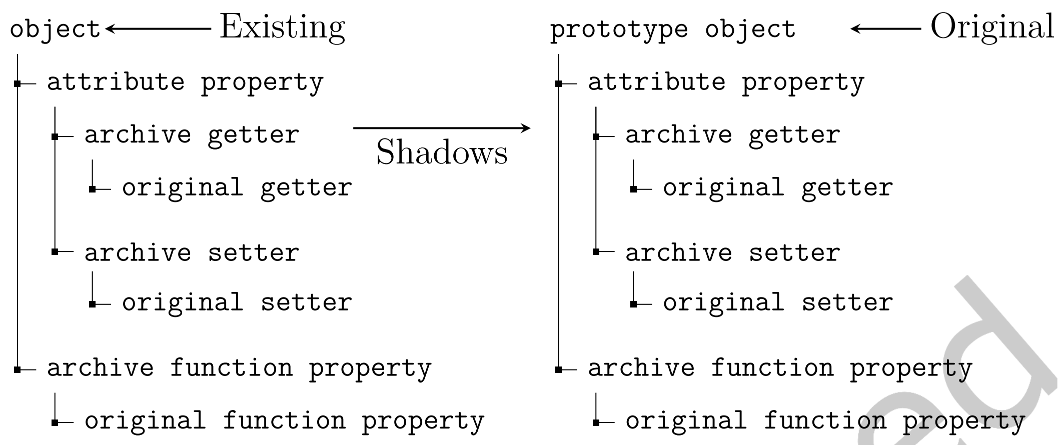
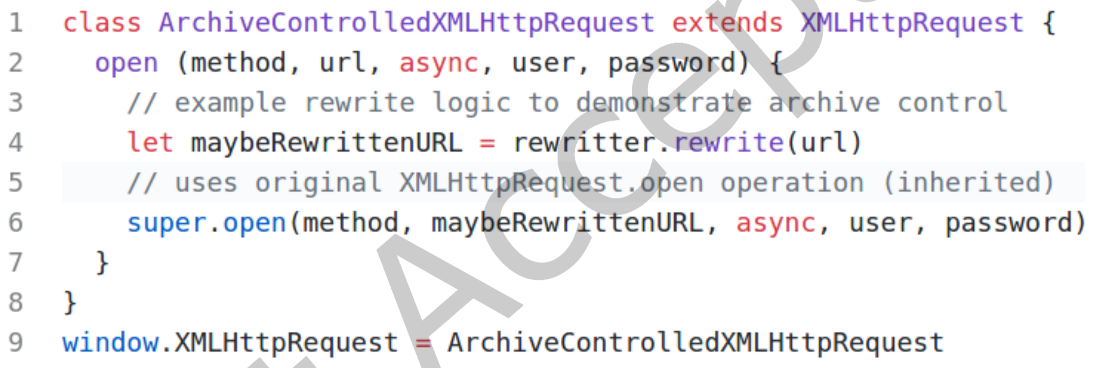


Fig. 25. Existing instance *replace plus patch* modification



```
class ArchiveControlledXMLHttpRequest extends XMLHttpRequest {
  open (method, url, async, user, password) {
    // example rewrite logic to demonstrate archive control
    let maybeRewrittenURL = rewritter.rewrite(url)
    // uses original XMLHttpRequest.open operation (inherited)
    super.open(method, maybeRewrittenURL, async, user, password)
  }
}
window.XMLHttpRequest = ArchiveControlledXMLHttpRequest
```

Fig. 26. Example *extend* override

global object `Window` and the `Document` interface (Figure 27). This override is the only one to introduce a new (foreign) representation of the interfaces it is targeting. Any assignment to the existing instance of the `Location` interface itself or to the interface's *href* attribute will navigate the browser away from the current page.

Because of the capabilities of the unforgeable `Location` interface, archives began rewriting server-side the text string "location" found in archived JavaScript to reference `WB_wombat_self_location`, an archive implementation of the `Location` interface. Even though server-side rewriting the text string "location" in the archived JavaScript of a page was more successful than server-side rewriting of URLs only, it was also rewriting instances of the text string "location" that were not actual instances of the `Location` interface.

Figure 28 shows an example of this incorrect rewriting. The live version of the page is shown in the right column, and the version replayed by Archive-It is shown in the left column. There are four instances of the term "location" that are rewritten to `WB_wombat_self_location`. But, this rewriting is only correctly done twice out of the four rewrites shown. The first incorrect rewrite happens for an object

```
[Unforgeable]
interface Location {
  attribute USVString href;
  void assign(USVString url);
  void replace(USVString url);
};

[PrimaryGlobal]
interface Window : EventTarget {
  [PutForwards=href, Unforgeable] readonly attribute Location? location;
};

interface Document : Node {
  [PutForwards=href, Unforgeable] readonly attribute Location? location;
}
```

Fig. 27. Foreign substitution unforgeable `Location` interface

that describes the location of the user (lines 2-12) and the second happens for the location property of an object that is the HTTP header of an HTTP response (lines 25-33). The remaining two rewrites (lines 18 and 19) were made correctly and are found in the code causing redirection when replayed because the page's JavaScript expects a cookie to exist which is nonexistent.

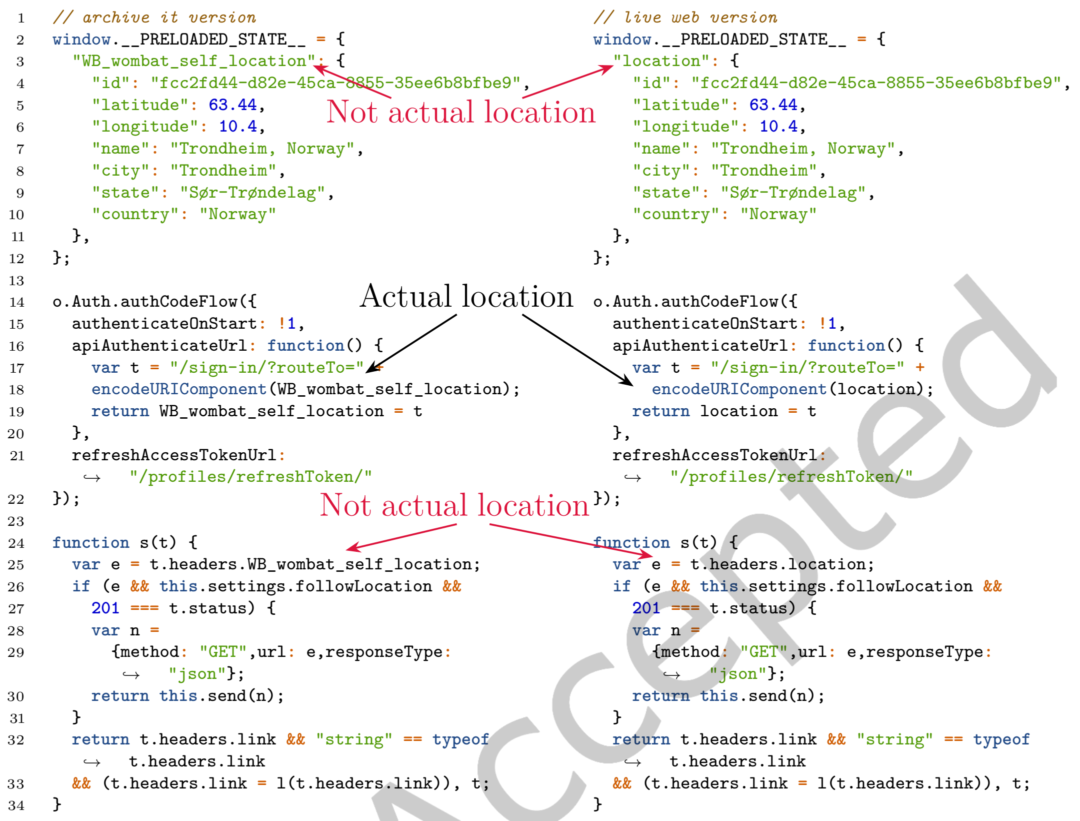
Webrecorder and Pywb, like Archive-It, were incorrectly rewriting the "location" text string in an archived page's JavaScript and also in non-JavaScript content bundled with the JavaScript. An example of this is shown in Figure 29. This page is from the documentation[21] for the ReactRouter JavaScript library, which bundles the example code for its documentation as HTML strings alongside the page's JavaScript. Because the example code was bundled with JavaScript, the text string "location" was incorrectly rewritten server-side due to its MIME type being *application/javascript* and not *text/html*.

To overcome this problem, we have developed a novel solution for eliminating the majority of additional rewrites required for the foreign substitution modification through the usage of a JavaScript `Proxy` object [Harband et al. 2021].

The JavaScript `Proxy` object allows an archive to perform runtime reflection for fundamental operations performed on or with the object being proxied. Simply put, the `Proxy` object allows for an archive to define custom behavior for all operations that can be performed with or on the object that cannot be done via the previous modifications via interceptors. This is especially useful for creating a more thorough override for both the `Window` and `Document` interfaces (Figure 30), both of which had properties that were incorrectly rewritten (Figure 28). As discussed previously, an archive cannot override the existing instance for the `Window` object, and likewise, an archive cannot directly proxy the existing instance for the `Window` interface but must proxy a plain object.

Because the archive proxy for the existing instance of the `Window` interface is in actuality a plain object, each new property addition intercepted by the window proxy must be added to both the plain object and the existing `Window` interface instance, as well the operation interceptors shown in Figure 30 lines 3-13. The existing instance for the `Document` interface on the other hand has no such restrictions, and

[21]https://reacttraining.com/react-router/web/example/auth-workflow



```
// archive it version
window.__PRELOADED_STATE__ = {
  "WB_wombat_self_location": {
    "id": "fcc2fd44-d82e-45ca-8855-35ee6b8bfbe9",
    "latitude": 63.44,
    "longitude": 10.4,
    "name": "Trondheim, Norway",
    "city": "Trondheim",
    "state": "Sør-Trøndelag",
    "country": "Norway"
  },
};

o.Auth.authCodeFlow({
  authenticateOnStart: !1,
  apiAuthenticateUrl: function() {
    var t = "/sign-in/?routeTo=" +
    encodeURIComponent(WB_wombat_self_location);
    return WB_wombat_self_location = t
  },
  refreshAccessTokenUrl:
   ↪   "/profiles/refreshToken/"
});

function s(t) {
  var e = t.headers.WB_wombat_self_location;
  if (e && this.settings.followLocation &&
    201 === t.status) {
    var n =
      {method: "GET",url: e,responseType:
       ↪   "json"};
    return this.send(n);
  }
  return t.headers.link && "string" == typeof
   ↪   t.headers.link
  && (t.headers.link = l(t.headers.link)), t;
}
```

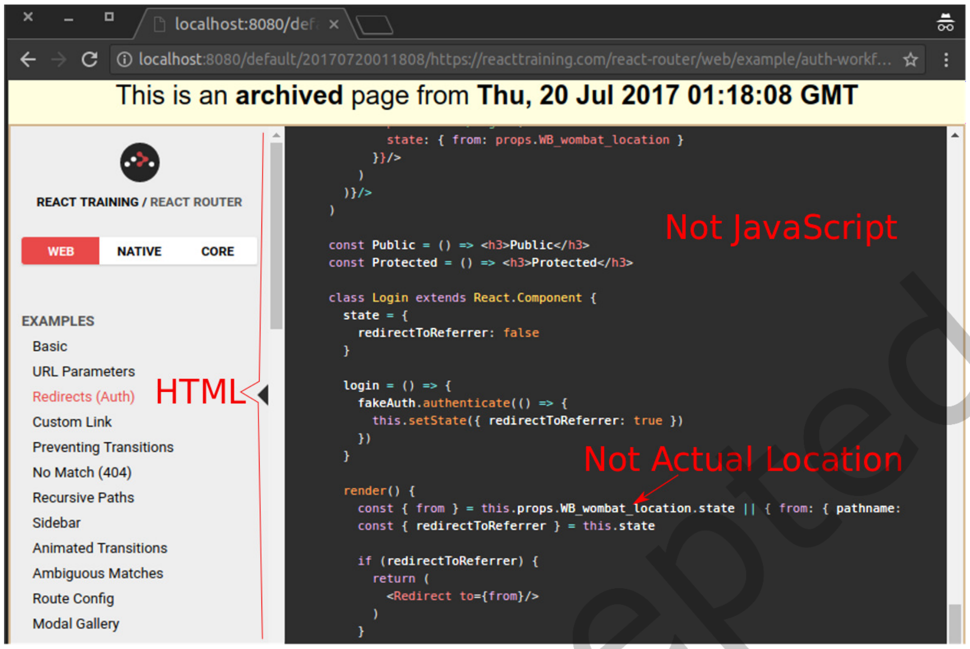
```
// live web version
window.__PRELOADED_STATE__ = {
  "location": {
    "id": "fcc2fd44-d82e-45ca-8855-35ee6b8bfbe9",
    "latitude": 63.44,
    "longitude": 10.4,
    "name": "Trondheim, Norway",
    "city": "Trondheim",
    "state": "Sør-Trøndelag",
    "country": "Norway"
  },
};

o.Auth.authCodeFlow({
  authenticateOnStart: !1,
  apiAuthenticateUrl: function() {
    var t = "/sign-in/?routeTo=" +
      encodeURIComponent(location);
    return location = t
  },
  refreshAccessTokenUrl:
   ↪   "/profiles/refreshToken/"
});

function s(t) {
  var e = t.headers.location;
  if (e && this.settings.followLocation &&
    201 === t.status) {
    var n =
      {method: "GET",url: e,responseType:
       ↪   "json"};
    return this.send(n);
  }
  return t.headers.link && "string" == typeof
   ↪   t.headers.link
  && (t.headers.link = l(t.headers.link)), t;
}
```

Fig. 28. Archive-It rewriting the text string "location" in the archived JavaScript of mendeley.com user pages. Live web version on the right.

archives can directly proxy the existing instance for the `Document` interface and need only to supply an interceptor for the property getter and setter (Figure 30 lines 18-19). This reduces the minimal server-side rewriting requirement for JavaScript to URL rewriting and the setup shown in Figure 31. By wrapping the archived JavaScript in an anonymous block scope and re-declaring each overridden interface's existing instance using the `let` declarator, an archive ensures that the archived JavaScript of page can only perform operations that the archive allows.

According to the compatibility tables for the `let` declarator[22] and JavaScript proxy object,[23] the minimal browser support for this method of performing the foreign substitution modification is Firefox v44, Chrome v49, Safari v10, Opera v36, and Microsoft Edge v12. This method for performing the foreign substitution modification and its initial implementation were contributed back to Pywb on April 28, 2017.[24] Both Webrecorder and Pywb have since fully adopted this method and has been using it in

[22]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let#browser_compatibility
[23]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy#browser_compatibility
[24]https://github.com/webrecorder/pywb/pull/215



```
          state: { from: props.WB_wombat_location }
        }}/>
      )
    )}/>
  )

  const Public = () => <h3>Public</h3>
  const Protected = () => <h3>Protected</h3>

  class Login extends React.Component {
    state = {
      redirectToReferrer: false
    }

    login = () => {
      fakeAuth.authenticate(() => {
        this.setState({ redirectToReferrer: true })
      })
    }

    render() {
      const { from } = this.props.WB_wombat_location.state || { from: { pathname:
      const { redirectToReferrer } = this.state

      if (redirectToReferrer) {
        return (
          <Redirect to={from}/>
        )
      }
```

Fig. 29. Pywb version 0.33 rewriting the text string "location" found in non-JavaScript page markup for the documentation of React Router. https://reacttraining.com/react-router/web/example/auth-workflow

production since August 21, 2017.[25] We include a description here so that it may be used by other archives to improve their replay fidelity and allow archived JavaScript to be replayed without modification.

5.2.4 Rewriter Generation. The overall process for generation of the client-side rewriter (Algorithm 5) uses **yield** to denote a function that generates JavaScript code and operates as follows. For each of the identified interfaces, if it inherits from `HTMLElement`, the *patch* override is generated for each of its attributes, and if the interface had a named constructor, then an *extend* override is generated. If the interface is a special check (Table 3), the overrides generated are determined by the interface's identifier (Algorithm 6). The `Window` interface generates the *replace* override, the `Document` interface generates the *replace plus patch* override, the `Location` interface has the *foreign substitution* override generated, and for all other special check interfaces, the *patch* override is generated. If the interface is neither an HTML element or a special check but is a function object, the *extend* override is generated. Otherwise, for each of its non-unforgeable attributes and operations, the *replace plus patch* override is generated (Algorithm 7).

6 EVALUATION

In this section, we evaluate the effectiveness of using a generalized client-side rewriter library to augment server-side rewriting. This would be applicable for archives that do not currently use client-side rewriting, such as the Internet Archive.

We formulate the following hypotheses:

[25]https://github.com/webrecorder/webrecorder/commit/12e2b507b88c4f0c00f29589436f7385dc512f9a

```
// window proxy
new window.Proxy({}, {
  get (target, prop) {/* intercept attribute getter calls */},
  set (target, prop, value) {/* intercept attribute setter calls */},
  has (target, prop) {/* intercept attribute lookup */},
  ownKeys (target) {/* intercept own property lookup */},
  getOwnPropertyDescriptor (target, key) {/* intercept descriptor lookup */},
  getPrototypeOf (target) {/* intercept prototype retrieval */},
  setPrototypeOf (target, newProto) {/* intercept changes of prototype */},
  isExtensible (target) {/* intercept is object extendable lookup */},
  preventExtensions (target) {/* intercept prevent extension calls */},
  deleteProperty (target, prop) {/* intercept is property deletion */},
  defineProperty (target, prop, desc) {/* intercept new property definition */},
})

// document proxy
new window.Proxy(window.document, {
  get (target, prop) {/* intercept attribute getter calls */},
  set (target, prop, value) {/* intercept attribute setter calls */}
})
```

Fig. 30. Archive `Window` and `Document` interface proxies

```
var __archive$assign$function__ = function(name) {/*return archive override*/};
{
  // archive overrides shadow these interfaces
  let window = __archive$assign$function__("window");
  let self = __archive$assign$function__("self");
  let document = __archive$assign$function__("document");
  let location = __archive$assign$function__("location");
  let top = __archive$assign$function__("top");
  let parent = __archive$assign$function__("parent");
  let frames = __archive$assign$function__("frames");
  let opener = __archive$assign$function__("opener");
  /* archived JavaScript */
}
```

Fig. 31. Archive JavaScript proxy setup anonymous block scope

H_1 Using client-side rewriting, the *number of requests made* by a composite memento *will increase* when replayed from the Wayback Machine, $ReqCS > Req$

H_2 Using client-side rewriting, the *number of requests blocked* by the content security policy of the Wayback Machine *will decrease*, $BlkReqCS < BlkReq$

H_3 Using client-side rewriting, the *number of requests made* by some composite mementos *will decrease* due to *archived JavaScript operating on rewritten URI-Rs* rather than un-rewritten URI-Rs, which are blocked and do not receive an HTTP response

Algorithm 5 Rewriter Generation

```
1: for each interface ∈ found do
2:     if inheritsFromHTMLElement(interface) then
3:         for each attr ∈ interface.attributes do
4:             yield PatchAttr(interface, attr)
5:         if interface.namedConstructor ≠ Nil then
6:             yield ExtendNamed(interface, interface.namedConstructor)
7:     else if isSpecialCheck(inteface) then
8:         yield GenerateSpecialCheck(inteface)
9:     else
10:        yield GenerateNoneElementNoneSpecialCheck(interface)
```

Algorithm 6 GenerateSpecialCheck

```
1: if interface.identifier == Window then
2:     for each attr ∈ interface.attributes do
3:         yield ReplaceAttr(interface,attr)
4:     for each operation ∈ interface.operations do
5:         yield ReplaceOperation(interface,operation)
6: else if interface.identifier == Document then
7:     for each attr ∈ interface.attributes do
8:         yield ReplacePlusPatchAttr(interface, attr)
9:     for each operation ∈ interface.operations do
10:        yield ReplacePlusPatchOperation(interface, operation)
11: else if interface.identifier == Location then
12:     yield ForeignSubstitution(interface)
13: else
14:     for each attr ∈ interface.attributes do
15:         yield PatchAttribute(interface, attr)
16:     for each operation ∈ interface.operations do
17:         yield PatchOperation(interface, operation)
```

Algorithm 7 GenerateNoneElementNoneSpecialCheck

```
1: if interface.constructor ≠ Nil ∨ interface.hadConstructor then
2:     yield Extend(interface)
3: else
4:     for each operation ∈ interface.operations do
5:         if !operation.unforgable then
6:             yield ReplacePlusPatchOperation(interface, operation)
7:     for each attribute ∈ interface.attributes do
8:         if !attribute.unforgable then
9:             yield ReplacePlusPatchAttribute(interface, attribute)
```

6.1 Data

Our goal was to evaluate the client-side rewriter on common cases, so we built a dataset containing popular webpages that were likely to be archived. This evaluation was carried out in late 2017, so we started with the June 2017 Alexa top 1,000,000 most visited websites.[26] We retrieved the TimeMaps for

[26]http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

Table 6. Crawler Recorded Metrics Term Definitions

Term	Definition
Req	Number of Requests Made, No Client-Side Rewriting
ReqCS	Number of Requests Made, With Client-Side Rewriting
RewrtCS	Number of Client-Side Rewrites
BlkReq	Number of Requests Blocked By CSP, No Client-Side Rewriting
BlkReqCS	Number of Requests Blocked By CSP, With Client-Side Rewriting

the top 3000 web pages (URI-Rs) from the list. From that, we selected the first 700 pages that had a memento in the Internet Archive. We wanted our study to reflect the state of the web page during June 2017 (when the site was in the top 3000 Alexa ranking), so we selected mementos from June 2017 if available. If a page did not have a memento between June 1 and June 30, 2017, we selected the latest memento (URI-M) available. We resolved each URI-M using Google Chrome, saving the final redirected URI-M for any pages that had an archived HTTP 3xx status. We discarded any URI-Ms that redirected more than 10 times or took longer than 20 seconds for the browser to render the page. This filtering resulted in a total of 577 URI-Ms to be evaluated.

The list of 577 URI-Ms along with the TLD count, Alexa rank, and temporal spread of the mementos is available at https://n0tan3rd.github.io/quickExploreCrawledData/. Most of the URI-Rs had a memento in June 2017, with only 8 URI-Rs having a latest memento before June 2017 (5 in 2013, 2 in 2016, and 1 in May 2017). Our dataset has pages from a wide range of Alexa rankings. The top ranked page was http://www.wikipedia.org (rank 5), and we had between 40-70 URI-Rs from each set of 100 ranks (i.e., ranks 1-99, 100-199, 200-299, ..., 900-999). The lowest ranked page in the dataset, http://www.umeng.com, had an Alexa rank of 2134.

6.2 Experiment

Once we had ensured all URI-Ms were resolved and navigable, we measured the difference in the number of requests made by the composite mementos from the Internet Archive's Wayback Machine with and without client-side rewriting.

Each page was crawled using the Google Chrome browser controlled using the Chrome DevTools Protocol[27] four times: twice without client-side rewriting and twice with client-side rewriting. For each crawl we recorded the number of requests made by the composite memento and the number of requests blocked by the Wayback Machine's CSP. The crawler also recorded each call to the console API. We recorded the calls to the console API because when the page was crawled with client-side rewriting and a rewrite occurred, the injected client-side rewriter logged each rewrite occurring client-side using the sentinel `REWRITE: un-rewritten-url -> rewritten-url` allowing us to measure the number of client-side rewrites. Table 6 defines the notation used throughout this evaluation when referring to crawler recorded metrics. The crawler visited each page for a maximum of 90 seconds or until network idle was determined. The determination for network idle was calculated by keeping track of the request and response pairs for a page, and when there was only one in-flight request (no response) for 3 seconds, the crawler moved to the next page.

[27]https://chromedevtools.github.io/devtools-protocol/

Table 7. Observed Request Increase, Rewrites Client-Side And Requests Blocked By CSP With and Without Client-Side Rewriting

Observed	**Mean**	**Median**	**Min**	**Max**
ΔReq	77	29	-7,984	3,766
$\Delta Req'$	87	39	-7,983	4,151
$RewrtCS$	233	27	0	15,096
$BlkReq$	11	4	0	405
$BlkReqCS$	1	0	0	144

To ensure an accurate count of the requests made by a page with and without client-side rewriting, we instructed the browser to inject JavaScript code when the document was loaded but before the embedded resources were evaluated. This instruction scrolled the page at one second intervals a maximum of 25 times or until the bottom of the page was reached. The only difference in the JavaScript code injected into each page was the inclusion or exclusion of the generated client-side rewriter. Also, to ensure an accurate count of the number of rewrites that occurred client-side, we configured the injected client-side rewriter to not rewrite URIs that were either already rewritten or were used internally by the Wayback Machine.

Once both sets of crawls had completed, we calculated ΔReq, the difference in the number of requests made by each composite memento with client-side rewriting (Equation 1), and $\Delta Req'$, the composite memento's difference in requests, not counting those blocked by the Wayback Machine's CSP (Equation 2).

$$\Delta Req = ReqCS - Req \tag{1}$$

$$\Delta Req' = ReqCS - BlkReqCS - Req - BlkReq \tag{2}$$

6.3 Results

Table 7 shows the increase in requests made possible with client-side rewriting. Recall that each additional request corresponds to a resource that was previously unable to be replayed from the Wayback Machine. There were on average 77 more requests made with client-side rewriting. Not counting requests blocked by the Wayback Machine's CSP, that number increased to 87 additional requests, as client-side rewriting prevented on average 10 requests ($\overline{BlkReq} - \overline{BlkReqCS}$) from being blocked by the CSP.

We also show the observed minimum and maximum increases in requests. There was at least one page for which client-side rewriting greatly increased the number requests made to the archive ($\Delta Req = 3{,}766$ and $\Delta Req' = 4{,}151$) and one page where client-side rewriting greatly reduced the number of requests made ($\Delta Req = \text{-}7{,}984$ and $\Delta Req' = \text{-}7{,}983$). Upon inspection of the instances where client-side rewriting reduced the number of requests made, we found that these pages were trying to set an HTTP cookie and then forcing a page reload. When the cookie was not presented in the next HTTP request (which would happen without client-side rewriting), the page would continue to try to set the cookie and reload. When client-side rewriting was applied, the cookie was able to be set and the page loaded normally, reducing the number of attempted reloads.

We note that the injection of the client-side rewriter did not occur for iframes created by JavaScript that set the value of the iframe's src attribute to "about:blank", because the Google Chrome browser

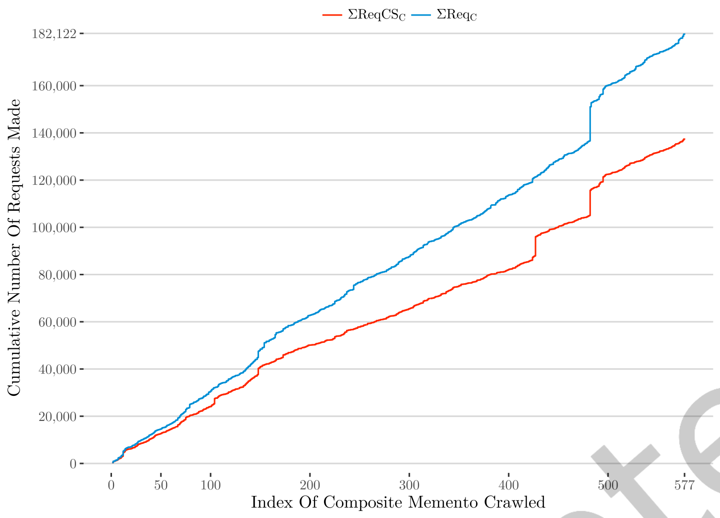
Table 8. Interface Operation Rewrites

Interface.operation	$RewrtCS$ **Count**
Element.getAttribute	12,109
Element.setAttribute	3,073
Document.write	3,030
Node.appendChild	865
Node.insertBefore	836
Node.replaceChild	120
Window.fetch	92
XMLHttpRequest.open	45
History.replaceState	33
Document.writeln	30
Element.insertAdjacentHTML	8
History.pushState	2

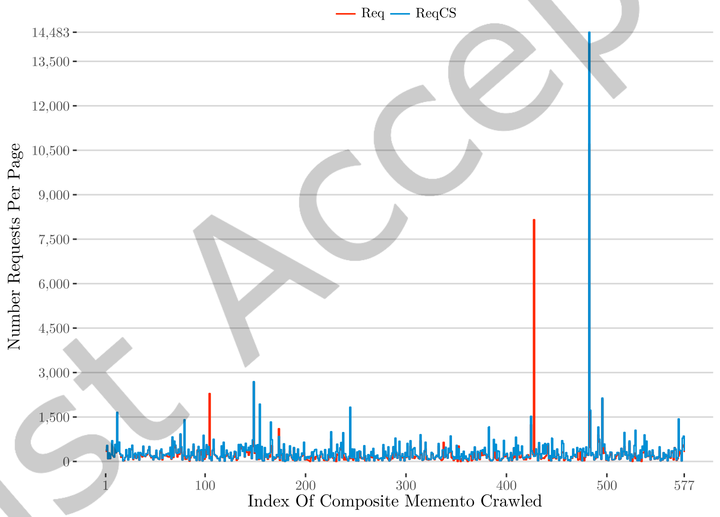
would only inject the code into browser contexts for a real origin. Even though the mean number of client-side rewrites per composite memento $\overline{RewrtCS}$ was 23, we observed that there were composite mementos crawled that did not require client-side rewriting at all, which is reflected by the observed minimum $BlkReq$ and $BlkReqCS$ values.

Similarly, the maximum $RewrtCS$ value observed ($RewrtCS = 15{,}096$) reflects that the crawler did visit a composite memento for which client-side rewriting greatly increased the number of requests made when replayed from the Wayback Machine. From numbers displayed in Table 7 it would appear that all three of our hypotheses (H_1, H_2, and H_3) are correct. To better understand the impact of client-side rewriting on replaying archived web pages via the Wayback Machine, consider Figure 32a, which displays the cumulative sum for the requests made with ($\Sigma ReqCS_{\mathcal{C}}$) and without ($\Sigma Req_{\mathcal{C}}$) client-side rewriting. The values for $\Sigma ReqCS_{\mathcal{C}}$ and $\Sigma Req_{\mathcal{C}}$ do not begin to diverge with any significance until the crawler had visited 80 pages (P_{80}), where we observed values $\Sigma Req_{\mathcal{C}} = 20{,}417$ and $\Sigma ReqCS_{\mathcal{C}} = 25{,}222$. At P_{200} we observed the values for $\Sigma Req_{\mathcal{C}}$ and $\Sigma ReqCS_{\mathcal{C}}$ start to diverge again, with $\Sigma Req_{\mathcal{C}} = 50{,}128$ and $\Sigma ReqCS_{\mathcal{C}} = 62{,}798$. It is not until P_{430} where we observed any significant increase in the values for $\Sigma Req_{\mathcal{C}}$ with $\Sigma Req_{\mathcal{C}} = 96{,}194$ and $\Sigma ReqCS_{\mathcal{C}} = 122{,}267$. At the end of the crawl, the total number of requests made without client-side rewriting ($\Sigma Req_{\mathcal{C}}$) was 137,071 and with client-side rewriting ($\Sigma ReqCS_{\mathcal{C}}$) was 182,122. As shown in Figure 32a, client-side rewriting increases the overall number of requests made by a page replayed from the Internet Archive's Wayback Machine. By the end of both crawls, the pages replayed with client-side rewriting made a total of 45,051 additional requests ($\Sigma ReqCS_{\mathcal{C}} - \Sigma Req_{\mathcal{C}}$), a 32.8% increase via 134,923 rewrites which occurred client-side (Figure 33a).

But before looking more closely at the decrease in the number of requests blocked by the Wayback Machine's CSP by using client-side rewriting, consider the breakdown of which of the identified interfaces were responsible for the rewrites (Figure 33a and Tables 8 and 9). Using the stack traces included with each of the console API's calls captured by the crawler, we were able to break down the originator of the rewrites into two categories identified as interface operations (Table 8) and general rewrites (Table 9).



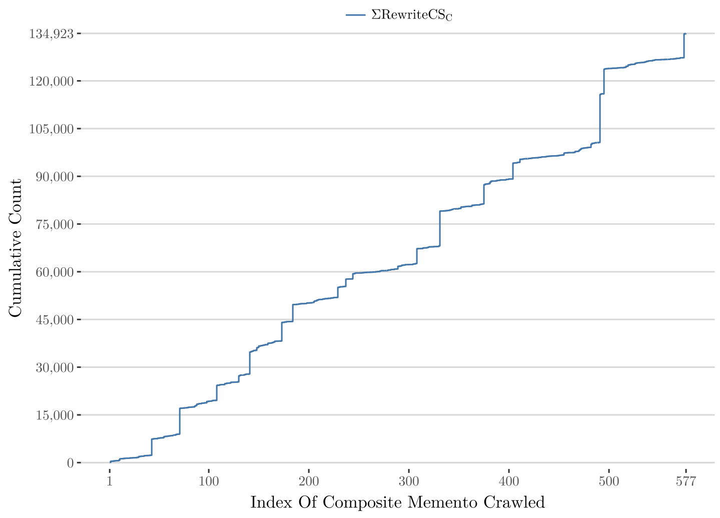
(a) $\Sigma Req_{\mathcal{C}}$ vs. $\Sigma ReqCS_{\mathcal{C}}$



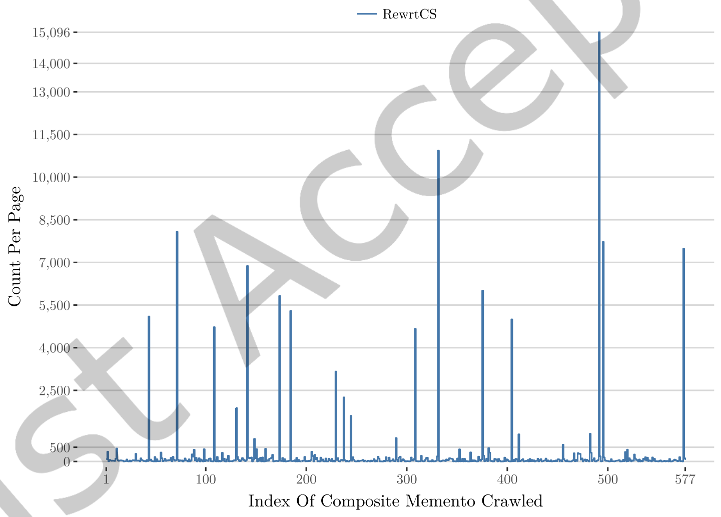
(b) Req vs. $ReqCS$ (per page)

Fig. 32. Cumulative number of requests (Figure 32a) and number of requests per page (Figure 32b) for 577 composite mementos replayed from the Internet Archive's Wayback Machine.

Because the majority of the archived JavaScript had undergone a minification process that obfuscated the function names, we were unable to concretely determine which of the identified interfaces were responsible for the rewrite. In those cases we use the name of the generated rewriter function which the rewrite originated from, namely `doRewrite` and `rewriteElement` (Table 9). The `doRewrite` function is the root function called for all rewrites and the `rewriteElement` function is responsible for rewriting instances of the `Element` and `Node` interfaces. The `getAttribute` ($RewrtCS = 12{,}109$) and `setAttribute`



(a) Cumulative number of client-side rewrites



(b) Number of client-side rewrites client-side per page.

Fig. 33. Cumulative number of client-side rewrites (Figure 33a) and number of client-side rewrites client-side per page (Figure 33b) for 577 composite mementos replayed from the Internet Archive's Wayback Machine.

($RewrtCS = 3{,}073$) operation of the `Element` interface and the `write` operation of the `Document` interface ($RewrtCS = 3{,}030$) were responsible for the majority of the rewrites originating from an operation of an identified interface (Table 8).

The operations of the `Node` interface, namely `appendChild`, `insertBefore`, and `replaceChild` were responsible for the majority of remaining rewrites (Table 8). Also of note, we were able to identify 92 rewrites occurring from the `fetch` operation of the `Window` interface, 45 rewrites

Table 9. General Rewrites

Rewrite Where	*RewrtCS* **Count**
doRewrite	94,975
rewriteElement	12,312
HTMLElement.style	2,381
HTML(Image|Source)Element.srcset	123

occurring from the `open` operation of the `XMLHTTPRequest` interface and 33 rewrites that occurred from the `replaceState` operation of the `History` interface. As previously mentioned, we were unable to concretely identify 94,975 rewrites (`doRewrite`) and 12,312 rewrites that occurred from the `rewriteElement` (Table 9). The remaining rewrites from the general rewrites category (Table 9) occurred from rewriting an instance of the `HTMLElement` interfaces `style` attribute and the `srcset` attribute from the `HTMLImageElement` or `HTMLSourceElement` interface.

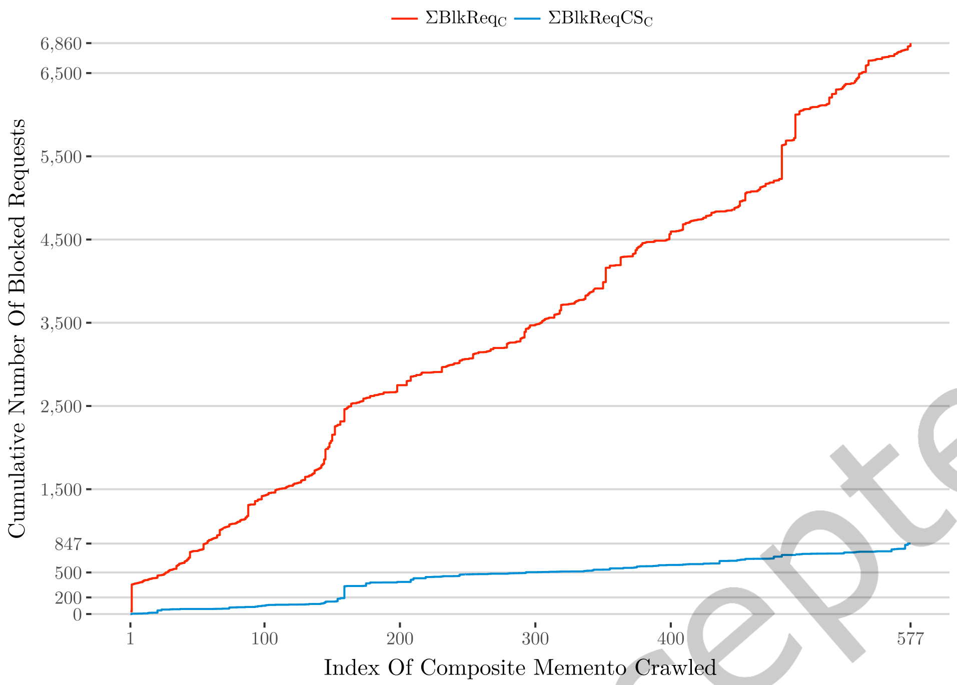
As shown by Tables 8 and 9, client-side rewriting would indeed increase the replay fidelity of the Internet Archive's Wayback Machine, with the increase in replay fidelity for the Wayback Machine becoming more clearly seen when considering the cumulative number of blocked requests with ($\Sigma BlkReqCS_{\mathcal{C}}$) and without ($\Sigma BlkReq_{\mathcal{C}}$) client-side rewriting, as shown in Figure 34a. At P_{100} we observed $\Sigma BlkReq_{\mathcal{C}} = 1,425$ requests were blocked by the content security policy of the Wayback Machine without client-side rewriting with only $\Sigma BlkReqCS_{\mathcal{C}} = 101$ requests blocked once client-side rewriting was applied. At P_{165} we observed the number of blocked requests for pages replayed with client-side rewriting increased sharply from $\Sigma BlkReqCS_{\mathcal{C}} = 184$, observed at P_{155} to $\Sigma BlkReqCS_{\mathcal{C}} = 337$. We also observed that between P_{100} and P_{165} the number of requests blocked for pages replayed without client-side nearly doubled from 1,425 to 2,533. After P_{165} $\Sigma BlkReqCS_{\mathcal{C}}$ slowly increased to 847 by the end of the crawl, whereas $\Sigma BlkReq_{\mathcal{C}}$ increased to $6,819$. Overall, this results in an decrease of 87.5%. As shown in Figure 34, the replay fidelity of the Internet Archive's Wayback Machine was increased by 5,972 requests ($\Sigma BlkReq_{\mathcal{C}} - \Sigma BlkReqCS_{\mathcal{C}}$) thus confirming H_2.

The third hypothesis H_3 is easily confirmed by considering Figure 35a, which shows the cumulative sum of the observed total increase with and without client-side rewriting for each page encountered in the Wayback Machine.

6.4 Revisiting the CNN Example

Recall the replay issue with the homepage of cnn.com discussed in Section 1. This page was crawled at P_{308} with $\Delta Req = 362$ when replayed with client-side rewriting. Because the generated client-side rewriting included an override for the `domain` attribute of the `Document` interface, the page was able to be replayed from the Internet Archive's Wayback Machine (Figure 36) in 2017, before the Internet Archive made their updates in 2020.

Even though the page is considered unreplayable (Figure 36b), 103 requests were made by the page and only three requests were blocked by the CSP of the Wayback Machine. But when the page is replayed with client-side rewriting (Figure 36a), 465 requests were made by the page, 0 requests were blocked by the CSP of the Wayback Machine, and 4,666 rewrites occurred client-side. By using client-side rewriting, the page's original issue was alleviated and allowed a 351% increase in requests made by the page's own JavaScript, effectively restoring the original behavior of the page.



(a) Cumulative number of blocked requests with and without client-side rewriting

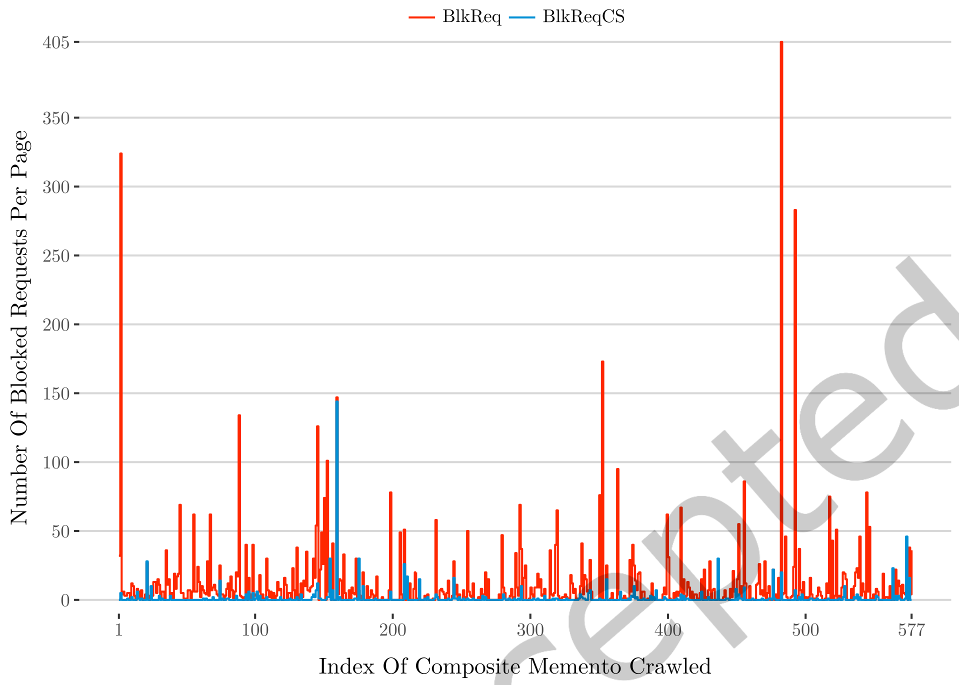
7 CONCLUSION

We have discussed in-depth how to securely replay archived JavaScript by proposing a framework for the automatic generation of client-side rewriting libraries. We defined a terminology for describing the modifications made by client-side rewriting libraries to the JavaScript execution environment of the browser. We also discussed how archives can reduce the amount of JavaScript rewriting required for facilitating client-side rewriting. We evaluated the effectiveness of client-side rewriting in augmenting the existing server-side rewriting systems of the Internet Archive.

Ensuring both high fidelity replay and the secure replay of archived JavaScript necessarily requires an archive to employ client-side rewriting. However, this requires archives to create their own client-side rewriting libraries and hand-tailor them to work with their existing server-side rewriting processes. To mitigate the time and labor intensive process of creating client-side rewriting libraries by hand, we proposed a framework for their auto-generation using the definitions of the targeted JavaScript APIs described in the Web Interface Design Language (Web IDL).

We showed how we can use the Web IDL definitions and the Web IDL to JavaScript mapping in combination with the specifications used by server-side rewriting, to generate a generic, archive independent, client-side rewriting library.

As shown by the evaluation of our proposed framework for the auto-generation of client-side rewriting libraries, client-side rewriting would both increase the replay fidelity of composite mementos and replay security of JavaScript from the Internet Archive's Wayback Machine. When the 577 composite mementos replayed from the Wayback Machine were crawled with client-side rewriting in 2017, we were able to decrease the total number of requests blocked by the content security policy of the Wayback Machine by



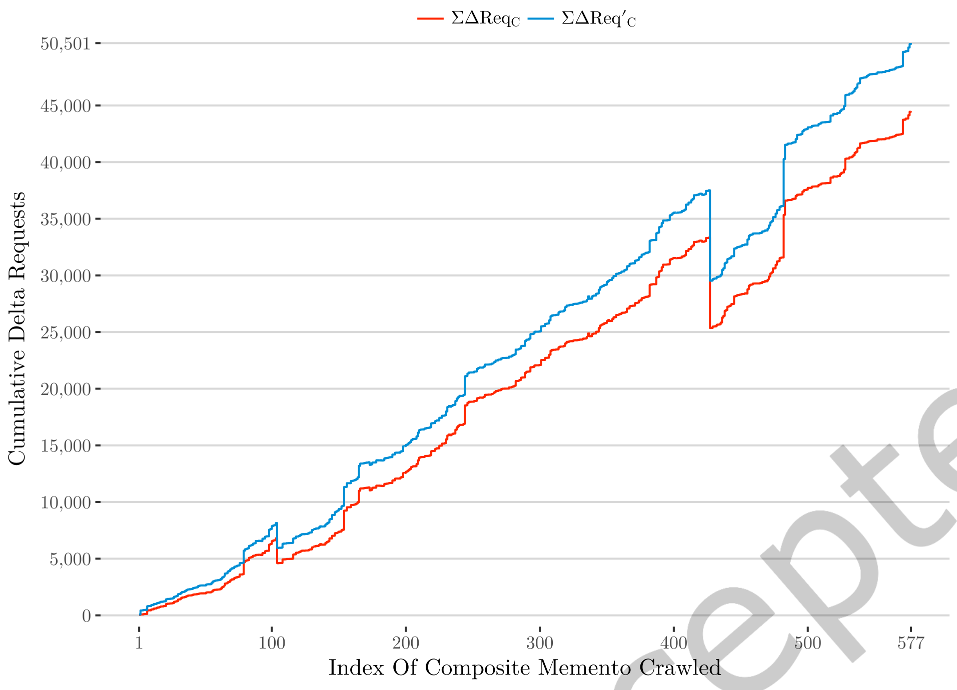
(b) Number of blocked requests with and without client-side rewriting per composite memento

Fig. 34. Cumulative number of blocked requests (Figure 34a) and number of blocked requests per page (Figure 34b) with and without client-side rewriting for 577 composite mementos replayed from the Internet Archive's Wayback Machine

87.5% and enabled an additional 45,051 requests to be made (replayed) by the composite mementos, an increase of 32.8%.

We suspected that we would see a decrease in the number of requests made by a composite memento because the archived JavaScript would be operating on URI-Ms rather than URI-Rs, which are blocked and do not receive an HTTP response. We also suspected that because the composite mementos' JavaScript was operating on URI-Rs and not URI-Ms, their replay would be impacted. This effect was due to the likelihood that the composite memento's JavaScript was configured to continually make requests for the same or fallback resource until a HTTP response is received. Our suspicions were confirmed when considering both the cumulative and per composite memento request deltas (Section 6.3), as the deltas revealed to us the extreme increases and decreases. We believe this would be a useful technique in identifying damaged mementos at scale.

Finally, as a direct result of including the generated the client-side rewriter in the replay of the composite mementos, we were able to make composite mementos that were previously un-replayable, replayable again. The home page of cnn.com became replayable again because the generated client-side rewriter applies an override targeting the document domain issue. Any page that also suffers from the document domain issue also becomes replayable when client-side rewriting is used that applies the necessary overrides for fixing that issue. Client-side rewriting solves the problem of variability in knowing



(a) $\Sigma\Delta Req_C$ vs. $\Sigma\Delta Req'_C$

which attributes are used by a page for embedding URLs, by applying overrides to the JavaScript APIs ultimately responsible for handling URLs.

This work has been adopted by the Webrecorder and Pywb playback systems and was used to address playback issues in the Internet Archive's Wayback Machine.

REFERENCES

Scott G. Ainsworth. 2015. Original Header Replay Considered Coherent. https://ws-dl.blogspot.com/2015/08/2015-08-28-original-header-replay.html.

Scott G. Ainsworth, Michael L. Nelson, and Herbert Van de Sompel. 2014. *A Framework for Evaluation of Composite Memento Temporal Coherence*. Technical Report arXiv:1402.0928. Old Dominion University.

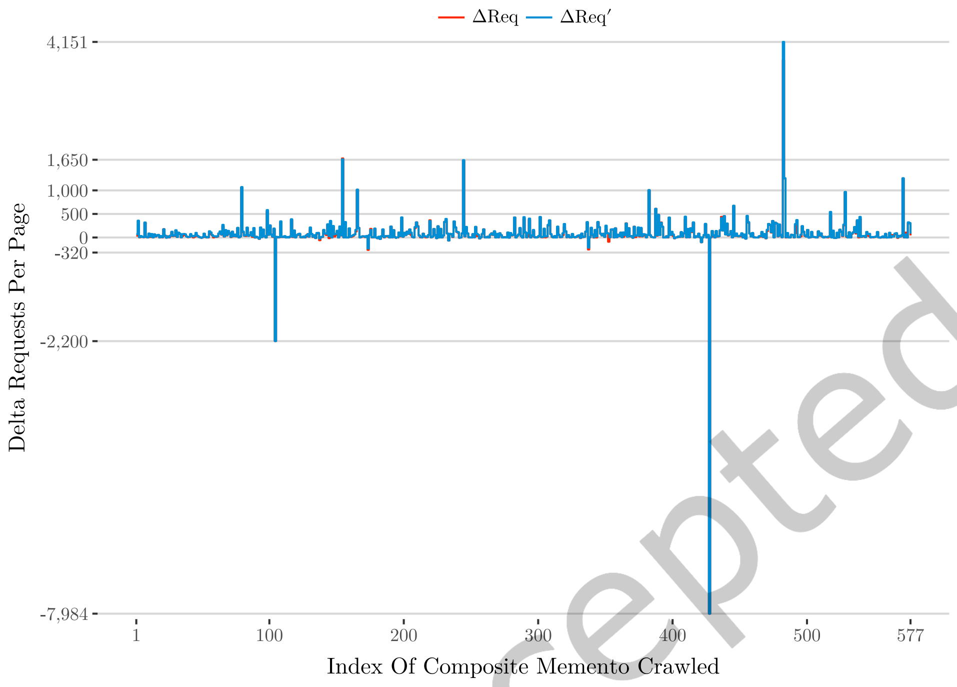
Scott G. Ainsworth, Michael L. Nelson, and Herbert Van de Sompel. 2015. Only One Out of Five Archived Web Pages Existed as Presented. In *Proceedings of the 26th ACM Conference on Hypertext & Social Media*. 257–266. https://doi.org/10.1145/2700171.2791044

Sawood Alam, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2017. Client-side Reconstruction of Composite Mementos Using ServiceWorker. In *Proceedings of the 17th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*. 1–4. https://doi.org/10.1109/JCDL.2017.7991579

Ahmed AlSum, Michele C. Weigle, Michael L. Nelson, and Herbert Van de Sompel. 2013. Profiling Web Archive Coverage for Top-Level Domain and Content Language. In *Proceedings of the International Conference on Theory and Practice of Digital Libraries (TPDL)*. 60–71. https://doi.org/10.1007/978-3-642-40501-3_7

Ahmed AlSum, Michele C. Weigle, Michael L. Nelson, and Herbert Van de Sompel. 2014. Profiling web archive coverage for top-level domain and content language. *International Journal on Digital Libraries (IJDL)* 14, 3 (2014), 149–166. https://doi.org/10.1007/s00799-014-0118-y

Mohamed Aturban, Michael L. Nelson, and Michele C. Weigle. 2021. Where Did the Web Archive Go?. In *International Conference on Theory and Practice of Digital Libraries (TPDL)*. Springer, 73–84. https://doi.org/10.1007/978-3-030-



(b) ΔReq vs. $\Delta Req'$ (per page)

Fig. 35. ΔReq and $\Delta Req'$ values for 577 composite mementos replayed from the Internet Archive's Wayback Machine

86324-1_9

Jefferson Bailey, Abigail Grotke, Kristine Hanna, Cathy Hartman, Edward McCain, Christie Moffatt, and Nicholas Taylor. 2013. Web Archiving in the United States: A 2013 Survey. https://blogs.loc.gov/thesignal/2014/10/results-from-the-2013-ndsa-u-s-web-archiving-survey/.

Vangelis Banos and Yannis Manolopoulos. 2016. A quantitative approach to evaluate Website Archivability using the CLEAR+ method. *International Journal on Digital Libraries (IJDL)* 17, 1 (2016), 119–141. https://doi.org/10.1007/s00799-015-0144-4

Adam Barth, Charles Reis, Collin Jackson, and Google Chrome Team. 2008. The Security Architecture of the Chromium Browser. https://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf.

John Berlin. 2017. CNN.com has been unarchivable since November 1st, 2016. https://ws-dl.blogspot.com/2017/01/2017-01-20-cnncom-has-been-unarchivable.html.

John Berlin. 2018. *To Relive The Web: A Framework for the Transformation and Archival Replay of Web Pages.* Master's thesis. Old Dominion University. https://digitalcommons.odu.edu/computerscience_etds/38/

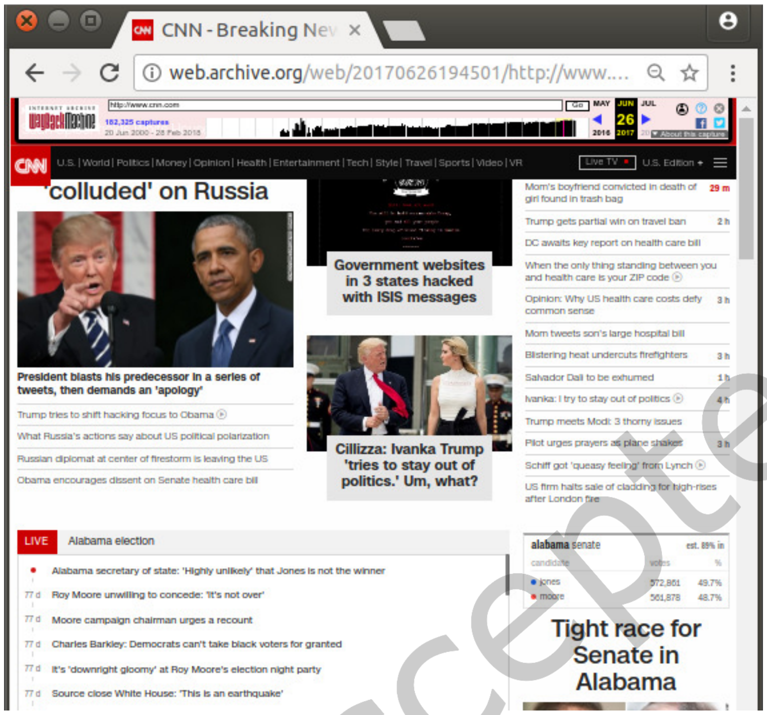
Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. 2005. *Uniform Resource Identifier (URI): Generic Syntax.* RFC 3986. https://www.rfc-editor.org/rfc/rfc3986.txt

Katherine E. Boss, Vicky Rampin, Remi Rampin, Fernando Chirigati, and Brian Hoffman. 2019. Saving Data Journalism: Using ReproZip-Web to Capture Dynamic Websites for Future Reuse. In *Proceedings of iPres*. 5. https://doi.org/10.31229/osf.io/khtdr

Niels Brügger. 2011. Web Archiving – between Past, Present, and Future. *The Handbook of Internet Studies* (2011), 24–42. https://doi.org/10.1002/9781444314861

Niels Brügger and Ian Milligan. 2018. *The SAGE Handbook of Web History.* SAGE. https://doi.org/10.4135/9781526470546

Justin F. Brunelle. 2012. Zombies in the Archives. https://ws-dl.blogspot.com/2012/10/2012-10-10-zombies-in-archives.html.


CNN - Breaking Nev
web.archive.org/web/20170626194501/http://www....
INTERNET ARCHIVE WayBackMachine
http://www.cnn.com
Go
182,325 captures
20 Jun 2000 - 28 Feb 2018
MAY JUN JUL
26
2016 2017
About this capture
U.S. | World | Politics | Money | Opinion | Health | Entertainment | Tech | Style | Travel | Sports | Video | VR
Live TV
U.S. Edition

'colluded' on Russia

President blasts his predecessor in a series of tweets, then demands an 'apology'

Trump tries to shift hacking focus to Obama

What Russia's actions say about US political polarization

Russian diplomat at center of firestorm is leaving the US

Obama encourages dissent on Senate health care bill

Government websites in 3 states hacked with ISIS messages

Cillizza: Ivanka Trump 'tries to stay out of politics.' Um, what?

Mom's boyfriend convicted in death of girl found in trash bag 29 m

Trump gets partial win on travel ban 2 h

DC awaits key report on health care bill

When the only thing standing between you and health care is your ZIP code

Opinion: Why US health care costs defy common sense 3 h

Mom tweets son's large hospital bill

Blistering heat undercuts firefighters 3 h

Salvador Dali to be exhumed 1 h

Ivanka: I try to stay out of politics 4 h

Trump meets Modi: 3 thorny issues

Pilot urges prayers as plane shakes 3 h

Schiff got 'queasy feeling' from Lynch

US firm halts sale of cladding for high-rises after London fire

LIVE Alabama election

Alabama secretary of state: 'Highly unlikely' that Jones is not the winner

77 d Roy Moore unwilling to concede: 'It's not over'

77 d Moore campaign chairman urges a recount

77 d Charles Barkley: Democrats can't take black voters for granted

77 d It's 'downright gloomy' at Roy Moore's election night party

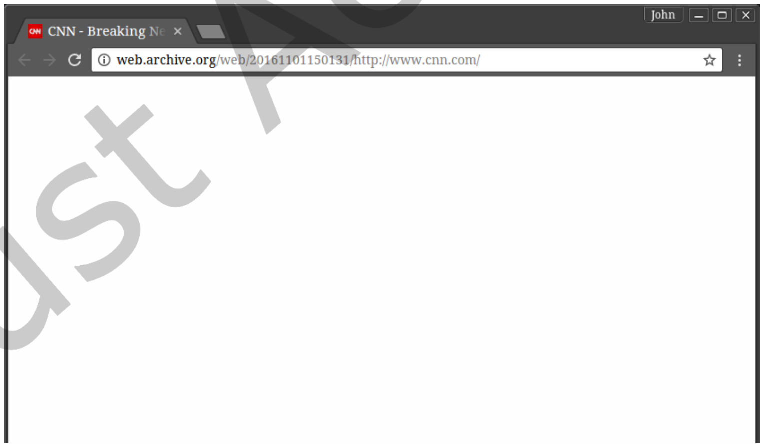
77 d Source close White House: 'This is an earthquake'

alabama senate est. 89% in

candidate	votes	%
● jones	572,861	49.7%
● moore	561,878	48.7%

Tight race for Senate in Alabama

(a) Home page of cnn.com replayed with client-side rewriting. http://web.archive.org/web/20170626194501/http://www.cnn.com (465 requests made, 0 blocked by CSP, 4,666 rewrites client-side)



(b) Home page of cnn.com replayed without client-side rewriting. http://web.archive.org/web/20170626194501/http://www.cnn.com (103 requests, 3 requests blocked by CSP)

Fig. 36. The home page of cnn.com is replayable from the Internet Archives Wayback Machine with client-side rewriting

Justin F. Brunelle, Mat Kelly, Hany SalahEldeen, Michele C. Weigle, and Michael L. Nelson. 2014. Not All Mementos Are Created Equal: Measuring the Impact of Missing Resources. In *Proceedings of ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. London, 321–330. https://doi.org/10.1109/JCDL.2014.6970187

Justin F. Brunelle, Mat Kelly, Hany SalahEldeen, Michele C. Weigle, and Michael L. Nelson. 2015. Not All Mementos Are Created Equal: Measuring the Impact of Missing Resources. *International Journal on Digital Libraries (IJDL)* 16 (May 2015), 283–301. Issue 3. https://doi.org/10.1007/s00799-015-0150-6

Justin F. Brunelle, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2016. The Impact of JavaScript on Archivability. *International Journal on Digital Libraries (IJDL)* 17, 2 (January 2016), 95–117. https://doi.org/10.1007/s00799-015-0140-8

Edgar Crook. 2009. Web archiving in a Web 2.0 world. *The Electronic Library* 27, 5 (2009), 831–836. https://doi.org/10.1108/02640470910998542

Jack Cushman. 2017. WARCgames. https://github.com/harvard-lil/warcgames.

Jack Cushman and Ilya Kreymer. 2017. Thinking like a hacker: Security Considerations for High-Fidelity Web Archives. Presented at International Internet Preservation Consortium (IIPC) Web Archiving Conference (WAC) 2017.

Deborah R. Eltgrowth. 2009. Best Evidence and the Wayback Machine: Toward a Workable Authentication Standard for Archived Internet Evidence. *Fordham Law Rev.* 78 (2009), 181.

Gunther Eysenbach and Mathieu Trudel. 2005. Going, Going, Still There: Using the WebCite Service to Permanently Archive Cited Web Pages. *Journal of Medical Internet Research* 7, 5 (2005), e920. https://doi.org/10.2196/jmir.7.5.e60

Matthew Farrell, Edward McCain, Maria Praetzellis, Grace Thomas, and Paige Walker. 2018. Web Archiving in the United States: A 2017 Survey. https://ndsa.org/2018/12/12/announcing-publication-of-ndsa-s-2017-web-archiving-survey-report.html.

Ian Fette and Alexey Melnikov. 2011. *The WebSocket Protocol*. RFC 6455. https://www.rfc-editor.org/rfc/rfc6455.txt

Sydney L. Forde, Robert E. Gutsche Jr, and Juliet Pinto. 2023. Exploring "ideological correction" in digital news updates of Portland protests & police violence. *Journalism* 24, 1 (2023), 157–176. https://doi.org/10.1177/14648849221100073

Robert Fox. 2001. Turning back 10 billion (web) pages of time. *Commun. ACM* 44 (2001), 9–10.

Lesley Frew. 2022. Web Archiving in Popular Media II: User Tasks of Journalists. https://ws-dl.blogspot.com/2022/08/2022-08-04-web-archiving-in-popular.html

Ayush Goel, Jingyuan Zhu, Ravi Netravali, and Harsha V. Madhyastha. 2022. Jawa: Web Archival in the Era of JavaScript. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 805–820. https://www.usenix.org/conference/osdi22/presentation/goel

Daniel Gomes, Elena Demidova, Jane Winters, and Thomas Risse. 2021. *The Past Web: Exploring Web Archives*. Springer. https://doi.org/10.1007/978-3-030-63291-5

Daniel Gomes, João Miranda, and Miguel Costa. 2011. A Survey on Web Archiving Initiatives. In *Proceedings of the International Conference on Theory and Practice of Digital Libraries (TPDL)*. 408–420. https://doi.org/10.1007/978-3-642-24469-8_41

Mark Graham. 2019. The Wayback Machine's Save Page Now is New and Improved. http://blog.archive.org/2019/10/23/the-wayback-machines-save-page-now-is-new-and-improved/.

Ilya Grigorik. 2018. Resource Hints. https://www.w3.org/TR/resource-hints/.

Jordan Harband, Shu yu Guo, Michael Ficarra, and Kevin Gibbons. 2021. ECMA-262, 12th edition, June 2021: ECMAScript® 2021 Language Specification. https://262.ecma-international.org/12.0/.

Helge Holzmann, Vinay Goel, and Avishek Anand. 2016. ArchiveSpark: Efficient Web Archive Access, Extraction and Derivation. In *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries (JCDL)*. 83–92. https://doi.org/10.1145/2910896.2910902

Helge Holzmann, Nick Ruest, Jefferson Bailey, Alex Dempsey, Samantha Fritz, Peggy Lee, and Ian Milligan. 2022. ABCDEF - The 6 key features behind scalable, multi-tenant web archive processing with ARCH: Archive, Big Data, Concurrent, Distributed, Efficient, Flexible. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 1–11.

International Internet Preservation Consortium (IIPC). 2015. OpenWayback. https://iipc.github.io/openwayback/2.1.0.RC.1/administrator_manual.html.

Internet Archive. 2022. News stories about the Internet Archive, filtered for "Wayback Machine". https://archive.org/about/news-stories/search?mentions-search=Wayback+Machine

Internet Archive Developer Portal. [n. d.]. Memento API. https://archive.readme.io/docs/memento.

ISO 28500. 2009. WARC (Web ARChive) file format. https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml.

Brewster Kahle. 2021. Reflections as the Internet Archive turns 25. https://blog.archive.org/2021/07/21/reflections-as-the-internet-archive-turns-25/.

Mat Kelly, Justin F. Brunelle, Michele C. Weigle, and Michael L. Nelson. 2013. On the Change in Archivability of Websites Over Time. In *Proceedings of the International Conference on Theory and Practice of Digital Libraries (TPDL)*. 35–47. https://doi.org/10.1007/978-3-642-40501-3_5

Mat Kelly, Michael L. Nelson, and Michele C. Weigle. 2014. The Archival Acid Test: Evaluating Archive Performance on Advanced HTML and JavaScript. In *Proceedings of the 14th IEEE/ACM Joint Conference on Digital Libraries (JCDL)*. 25–28. https://doi.org/10.1109/JCDL.2014.6970146

Martin Klein, Harihar Shankar, Lyudmila Balakireva, and Herbert Van de Sompel. 2019. The Memento Tracer Framework: Balancing Quality and Scalability for Web Archiving. In *Proceedings of the International Conference on Theory and Practice of Digital Libraries (TPDL)*. 163–176. https://doi.org/10.1007/978-3-030-30760-8_15

Ilya Kreymer. 2013. PyWb - Web Archiving Tools for All. https://github.com/ikreymer/pywb.

Ilya Kreymer. 2019. Wombat. https://github.com/webrecorder/wombat.

Ilya Kreymer. 2020. A New Phase for Webrecorder Project, Conifer and ReplayWeb.page. https://webrecorder.net/2020/06/11/webrecorder-conifer-and-replayweb-page.html.

Adam Kriesberg and Amelia Acker. 2022. The second US presidential social media transition: How private platforms impact the digital preservation of public records. *Journal of the Association for Information Science and Technology* 73 (2022), 1529–1542. Issue 11. https://doi.org/10.1002/asi.24659

Kalev Leetaru. 2015. How Much Of The Internet Does The Wayback Machine Really Archive? https://www.forbes.com/sites/kalevleetaru/2015/11/16/how-much-of-the-internet-does-the-wayback-machine-really-archive/#1f64c0679446.

Kalev Leetaru. 2017. Are Web Archives Failing The Modern Web: Video, Social Media, Dynamic Pages and The Mobile Web. https://www.forbes.com/sites/kalevleetaru/2017/02/24/are-web-archives-failing-the-modern-web-video-social-media-dynamic-pages-and-the-mobile-web/.

Ada Lerner, Tadayoshi Kohno, and Franziska Roesner. 2017. Rewriting History: Changing the Archived Web from the Present. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'17)*. 1741–1755. https://doi.org/10.1145/3133956.3134042

Jimmy Lin, Ian Milligan, Douglas W. Oard, Nick Ruest, and Katie Shilton. 2020. We Could, but Should We?: Ethical Considerations for Providing Access to GeoCities and Other Historical Digital Collections. In *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval*. 135–144. https://doi.org/10.1145/3343413.3377980

Daniela Major and Daniel Gomes. 2021. *Web Archives Preserve Our Digital Collective Memory*. Springer International Publishing, 11–19. https://doi.org/10.1007/978-3-030-63291-5_2

Julien Masanès. 2006. Web Archiving: Issues and Methods. In *Web Archiving*. Springer, 1–53. https://doi.org/10.1007/978-3-540-46332-0_1

Ian Milligan. 2019. *History in the Age of Abundance?: How the Web Is Transforming Historical Research*. McGill-Queen's Press-MQUP.

Gordon Mohr, Michael Stack, Igor Ranitovic, Dan Avery, and Michele Kimpton. 2004. An Introduction to Heritrix, an open source archival quality web crawler. In *4th International Web Archiving Workshop (IWAW'04)*. 109–115.

Michael L. Nelson. 2013a. Archive.is Supports Memento. https://ws-dl.blogspot.com/2013/07/2013-07-09-archiveis-supports-memento.html.

Michael L. Nelson. 2013b. Game Walkthroughs As A Metaphor for Web Preservation. https://ws-dl.blogspot.com/2013/05/2013-05-25-game-walkthroughs-as.html.

Michael L. Nelson. 2014. "Refresh" For Zombies, Time Jumps. https://ws-dl.blogspot.com/2014/07/2014-07-14-refresh-for-zombies-time.html.

Michael L. Nelson. 2020. At the nexus of the CNI keynote and Rosenthal's response: "It's not an easy thing to meet your maker.". https://ws-dl.blogspot.com/2020/03/2020-03-07-at-nexus-of-cni-keynote-and.html.

Mark Nottingham. 2014. *URI Design and Ownership*. RFC 7320. https://www.rfc-editor.org/rfc/rfc7320.txt

James L. Quarles III and Richard A. Crudo. 2014. [Way]Back to the Future: Using the Wayback Machine in patent litigation. *Landslide Magazine* 6, 3 (2014), 16.

Charles Reis, Adam Barth, and Carlos Pizano. 2009. Browser security: Lessons from Google Chrome. *Commun. ACM* 52, 8 (2009), 45–49. https://doi.org/10.1145/1536616.1536634

Brenda Reyes Ayala. 2022. Correspondence as the primary measure of information quality for web archives: A human-centered grounded theory study. *International Journal on Digital Libraries (IJDL)* 23, 1 (2022), 19–31. https://doi.org/10.1007/s00799-021-00314-x

David S. H. Rosenthal. 2012. Harvesting and Preserving the Future Web. https://blog.dshr.org/2012/05/harvesting-and-preserving-future-web.html.

David S. H. Rosenthal. 2017. Security Issues for Web Archives. https://blog.dshr.org/2017/06/wac2017-security-issues-for-web-archives.html.

Nick Ruest, Jimmy Lin, Ian Milligan, and Samantha Fritz. 2020. The Archives Unleashed Project: Technology, Process, and Community to Improve Scholarly Access to Web Archives. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 157–166. https://doi.org/10.1145/3383583.3398513

Tim Sherratt and Andrew Jackson. 2020. GLAM-Workbench/web-archives. https://doi.org/10.5281/zenodo.3894079.

Hunter Stern. 2011. Fetch Chain Processors. https://webarchive.jira.com/wiki/display/Heritrix/Fetch+Chain+Processors.

Brad Tofel. 2007. Wayback for Accessing Web Archives. In *7th International Web Archiving Workshop (IWAW'07)*. 27–37.

Masashi Toyoda and Masaru Kitsuregawa. 2012. The History of Web Archiving. In *Proceedings of the IEEE*, Vol. 100. IEEE, 1441–1443. https://doi.org/10.1109/JPROC.2012.2189920 Special Centennial Issue.

Herbert Van de Sompel, Michael Nelson, and Robert Sanderson. 2013. *HTTP Framework for Time-Based Access to Resource States – Memento*. RFC 7089. https://www.rfc-editor.org/rfc/rfc7089.txt

Herbert Van de Sompel, Michael L. Nelson, Robert Sanderson, Lyudmila L. Balakireva, Scott Ainsworth, and Harihar Shankar. 2009. *Memento: Time Travel for the Web*. Technical Report arXiv:0911.1112.

Anne van Kesteren. 2016. Defining the WindowProxy, Window, and Location objects. https://blog.whatwg.org/windowproxy-window-and-location.

Anne van Kesteren. 2020. Cross-Origin Resource Sharing. https://www.w3.org/TR/2020/SPSD-cors-20200602/.

W3C. 2022. Cascading Style Sheets. https://www.w3.org/Style/CSS/Overview.en.html.

Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Tatsuya Mori. 2020. Melting Pot of Origins: Compromising the Intermediary Web Services that Rehost Websites. In *Network and Distributed Systems Security (NDSS) Symposium*. 15. https://doi.org/10.14722/ndss.2020.24140

Michele C. Weigle. 2022. Using Web Archives in Disinformation Research. https://ws-dl.blogspot.com/2022/09/2022-09-28-using-web-archives-in.html

Joel Weinberger, Frederik Braun, Devdatta Akhawe, and Francois Marier. 2016. Subresource Integrity. https://w3c.github.io/webappsec-subresource-integrity/.

Mike West, Adam Barth, and Dan Veditz. 2016. Content Security Policy Level 2. https://www.w3.org/TR/CSP2/.

WHATWG Working Group. 2017a. DOM Living Standard. https://dom.spec.whatwg.org/.

WHATWG Working Group. 2017b. WebIDL Level 1. https://www.w3.org/TR/WebIDL-1/.

WHATWG Working Group. 2022. HTML Living Standard. https://html.spec.whatwg.org/.

Jonathan Zittrain, Kendra Albert, and Lawrence Lessig. 2014. Perma: Scoping and addressing the problem of link and reference rot in legal citations. *Legal Information Management* 14, 2 (2014), 88–99. https://doi.org/10.1017/S1472669614000255