

**TO RELIVE THE WEB: A FRAMEWORK FOR THE
TRANSFORMATION AND ARCHIVAL REPLAY OF
WEB PAGES**

by

John Andrew Berlin
B.S. December 2015, Old Dominion University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 2018

Approved by:

Michele C. Weigle (Director)

Michael L. Nelson (Member)

Justin F. Brunelle (Member)

ABSTRACT

TO RELIVE THE WEB: A FRAMEWORK FOR THE TRANSFORMATION AND ARCHIVAL REPLAY OF WEB PAGES

John Andrew Berlin
Old Dominion University, 2018
Director: Dr. Michele C. Weigle

When replaying an archived web page (known as a 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 to modify the page and its embedded resources, so that they no longer reference (link to) the original server(s) they were archived from but back to the archive. 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. Unfortunately, because the replay of mementos and the modifications made to them by web archives in order to facilitate replay varies between archives, the terminology for describing replay and the modification made to mementos for facilitating replay does not exist. In this thesis, we propose terminology for describing the existing styles of replay and the modifications made on the part of web archives to mementos in order to facilitate replay. This thesis also, in the process of defining terminology for the modifications made by client-side rewriting libraries to the JavaScript execution environment of the browser during replay, proposes 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%. Also by using the generated client-side rewriter, we were able to replay mementos that were previously not replayable from the Internet Archive.

Copyright, 2018, by John Andrew Berlin, All Rights Reserved.

ACKNOWLEDGEMENTS

First I would like to thank God who ultimately gave me the strength and ability to complete this thesis.

This thesis would not have been possible without the guidance, support, and seemingly endless patience of my advisers Dr. Michele C. Weigle and Dr. Michael Nelson. Without it, I would have been forever lost in a sea of JavaScript with no chance of coming up for air and have learned so much from their tutelage for which I am eternally grateful. I would like to thank Dr. Nelson for pushing me to investigate the replay failure of `cnn.com` and the forever redirecting `mendely.com` user page as these two investigations laid the foundation for this thesis topic. I am also grateful to Dr. Weigle for giving me the opportunity while I was still an undergraduate to work on a WSDL project which opened the door for me to join the research group.

I would like to give a huge shout out to Mat Kelly for being the creator of the web archiving projects I cut my teeth on. If he had not created WAIL, Mink, and WARCreate, I would not have had a starting point for all things JavaScript. Another shout out goes to Sawood Alam for being an excellent sounding board for ideas and always playing devils advocate.

Finally I am extremely grateful to my family, especially my grandparents, for their encouragement and support throughout the process of completing this thesis.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	xv
Chapter	
1. INTRODUCTION	1
1.1 CONTRIBUTIONS	11
1.2 THESIS ORGANIZATION	11
2. BACKGROUND	13
2.1 HYPER TEXT TRANSPORT PROTOCOL (HTTP)	13
2.2 HYPER TEXT MARKUP LANGUAGE	17
2.3 JAVASCRIPT (JS)	20
2.4 BROWSERS AND VIEWING A WEBPAGE	23
2.5 MEMENTO FRAMEWORK	26
2.6 ARCHIVING AND REPLAY OF WEB PAGES	28
2.7 URL REWRITING	32
2.8 SUMMARY	36
3. RELATED WORK	37
3.1 ARCHIVING DYNAMIC CONTENT	37
3.2 HANDLING THE REPLAY OF DYNAMIC CONTENT	38
3.3 SECURITY AND THE ARCHIVE	39
3.4 SUMMARY	40
4. STYLES OF REPLAYING ARCHIVED WEB PAGES	41
4.1 ARCHIVAL LINKAGE MODIFICATIONS	41
4.2 REPLAY PRESERVING MODIFICATIONS	47
4.3 SANDBOXED REPLAY	54
4.4 NON-SANDBOXING REPLAY	65
4.5 ESSENCE PRESERVATION	71
4.6 SUMMARY	85
5. JAVASCRIPT FOR THE PRESERVATION AND REPLAY OF THE MOD- ERN WEB	88
5.1 WEB IDL	89
5.2 WEB IDL JAVASCRIPT MAPPING	91
5.3 AUTO-GENERATING A CLIENT-SIDE REWRITER	94
5.4 EVALUATION	148
5.5 WAYBACK MACHINE BANNER VULNERABILITY	174

5.6 SUMMARY	180
6. CONTRIBUTIONS, FUTURE WORK, AND CONCLUSIONS.....	184
6.1 CONTRIBUTIONS	184
6.2 FUTURE WORK	185
6.3 CONCLUSIONS	185
REFERENCES.....	188
APPENDICES	
A. JAVASCRIPT FOR THE PRESERVATION AND REPLAY OF THE MODERN WEB	195
VITA.....	204

LIST OF TABLES

Table	Page
1. RFC 7231 HTTP methods	15
2. HTTP status code classes	16
3. HTML elements containing the src and href attributes	42
4. Link Rel Types Requiring A Browser Resource Fetch	43
5. Content Security Policy Directives	54
6. Terminology for describing the “Wayback” model of replay	85
7. Terminology describing the modifications made to mementos to facilitate replay using the “Wayback” model	86
8. Terminology for describing the replay of and modifications made to me- mementos by the “Non-Wayback” model	87
9. HTML Element Attributes With Rewrite Modifier From Pywb	96
10. Baseline Interface Discovery Identifiers	121
11. Special Well-Known Interface and Member Identifiers	121
12. Identified HTML Interfaces	126
13. Identified Non-Element or Special Case Interfaces	127
14. Crawler Recorded Metrics Term Definitions	151
15. Observed Request Increase, Rewrites Client-Side And Requests Blocked By CSP With and Without Client-Side Rewriting	152
16. Interface Operation Rewrites	157
17. General Rewrites	157
18. Terms describing the modifications made to the JavaScript execution environment of the browser by client-side rewriting libraries	182

LIST OF FIGURES

Figure	Page
1. Replay failure of the home page for CNN.com	4
3. Replay errors for the home page of CNN as seen by the developer console	5
4. archive.is archival transformation of CNN	6
5. Replay failures of <code>https://www.mendeley.com/profiles/helen-palmer</code>	9
6. Archive-It rewriting the identifier “Location” in the archived JavaScript of mendeley.com user pages	10
7. Anatomy of a URI	14
8. Anatomy of content negotiation	15
9. HTTP request with redirection	17
10. The HTML of the famous <code>http://example.com</code>	19
11. <code>http://example.com</code> rendered by Google Chrome	20
12. JavaScript making an HTTP request for an image and mutating the DOM by adding an <code>img</code> tag whose <code>src</code> attribute is a Blob URL	21
13. Ways additional JavaScript can be introduced into the page	21
14. JavaScript adding an HTTP cookie to the page	22
15. JavaScript changing the location of the browser	22
16. JavaScript for changing browser history	23
17. JavaScript for changing the domain of the browser	23
18. Anatomy of a CORS request	25
19. Temporal content negotiation with the Internet Archive’s TimeGate to select the best Memento for the web page <code>http://grindtodeath.com/</code> at the datetime Wed, 31 Oct 2012 09:52:40 GMT	26
20. TimeMap for the web page <code>http://grindtodeath.com/</code> retrieved from the Internet Archive	27

21.	Wayback Machine's interface For http://www.nocleansinging.com/	31
22.	WARC file contents containing the HTTP requests and responses for the preserved web page http://example.com	32
23.	Pre And Post Rewritten Link Tags	33
24.	Pywb CSS regex rewriter	34
25.	URIs in CSS files	35
26.	OpenWayback standard attribute rewrite configuration	35
27.	Archival Acid Test tool evaluation results as of December 2014, [45, p. 3 Table 2]	37
28.	Custom HTML element using custom attributes	44
29.	Live web JavaScript usage of URI-Rs	45
30.	Script tag embedding JSON	46
31.	Request made to Flickr API using embedded JSON	46
32.	Response body for the GET request to Flickr API	46
33.	URIs in CSS files	47
34.	Meta refresh tag	47
35.	Content Security Policy defined in a meta tag	48
36.	The HTML of the page delivering a Content Security Policy via meta tag with the areas affected by the policy during replay from the Internet Archive's Wayback Machine highlighted	50
37.	Meta Tag Delivered Content Security Policy preventing Internet Archive From Loading Wayback Machine Resources	51
38.	Blocked Wayback and archived embedded resources	52
39.	Meta tag delivered content security policy preventing Internet Archive from controlling save page now	53
40.	Integrity attribute of the script and link tags	54
41.	Example of replay isolation's sandbox on a page replayed by webrecorder.io. The sandboxed portion is outlined in red.	56

42.	Sandboxing Replay frame tree. The green box represents the webrecorder.io domain and the red box represents the webrc.io domain	58
43.	Replay Isolation user view. The green box represents the necessarily archive controlled portion of replay and the red box represents the replayed page.	59
44.	Ways in which JavaScript may detect it is being replayed inside an iframe	60
45.	Frame busting JavaScript Github search	61
46.	Location exposing and navigation control JavaScript APIs	62
47.	Replay Isolation location override implementation	63
48.	Direct function overriding of JavaScript HTTP request APIs	64
49.	Direct element attribute and content rewriting.	65
50.	Example of non-sandboxing replay. The replayed page's contents are outlined in red.	66
51.	Non-Sandboxing replay frame tree	68
52.	Non-Sandboxing replay user view. The green box represents the necessarily archive control portion of replay and the red box represents the replayed page	69
53.	<code>http://example.com</code> as it exists on the live web	70
54.	Internet Archive's injected banners vulnerability to a memento embedded CSS due to non-sandboxing replay	70
55.	Internet Archive's injected banners vulnerability (Figure 54), offending <i>div</i> and memento's embedded CSS	71
56.	PastPages preserving news sites essence through images.	72
57.	Example <code>http://cs.odu.edu/~jberlin/originalThreeGreen.html</code>	74
58.	Simple example of Archival Caricaturization preserving exactly how the page existed. Rendering of HTML shown in Figure 57.	75
59.	Transformation of HTML shown in Figure 57 as archived through caricaturization	76
60.	archive.is caricaturization of <code>nocleansing.com</code> , the live web version is on the left, and the memento is on the right	77

61.	heavyblogisheavy.com vs. archive.is no zoom	78
62.	heavyblogisheavy.com vs. archive.is zoom 50%	79
63.	heavyblogisheavy.com vs. archive.is zoom 90%	79
64.	archive.is addition of display:none to heavyblogisheavy.com.....	81
65.	Page using JavaScript powered custom HTML elements as the primary markup	83
66.	Page using JavaScript powered custom HTML elements as the primary markup as replayed from archive.is demonstrating the inability of archive.is to correctly preserve the page through caricature	84
67.	Web IDL syntax.....	90
68.	Web IDL extended attributes and typedefs	90
69.	Unforgeable and NoInterfaceObject extended attributes.....	92
70.	Web IDL interface to ECMAScript mapping	93
71.	HTMLAnchorElement.idl	94
72.	CSS style properties that may contain URLs and how URLs may exist in CSS style definitions found in a style tag.....	97
73.	CSS in Web IDL	98
74.	Dynamic changes to the <i>textContent</i> attribute of a style tag to load URLs live web	99
75.	Dynamic changes to the <i>textContent</i> attribute of a style tag to load URLs archived	100
76.	Well-known property mutating identifiers	101
77.	Well-known document mutating identifiers	102
78.	DOMParser Web IDL Interface	102
79.	bbc.com new story footer images blocked by the Wayback Machine's content security policy	104
80.	bbc.com new story footer images blocked by the Wayback Machine's content security policy, network tab of browser developer tools.....	105

81.	bbc.com new story footer blocked images <code>img</code> tag CSS classes	105
82.	Embedded configuration for lazy loading the additional stories images . . .	106
83.	Portion of the response to the request for the additional stories images “/news/pattttern-library-components?...”	106
84.	bbc.com lazy loading code	107
85.	http://www.soufeel.com/ archived vs live web	109
86.	Network tab of the developers console when replaying soufeel.com from the Wayback Machine displaying the blocked images	110
87.	Select HTML elements used by soufeel.com in lazy loading way 1.	110
88.	soufeel.com lazy loading way 1, code embedded in HTML and formatted for presentation	111
89.	soufeel.com select HTML elements used by lazy loading way 2. Formatted for presentation	112
90.	soufeel.com lazy loading way 2 code. Formatted for presentation	112
91.	soufeel.com select HTML elements used by lazy loading way 3	113
92.	soufeel.com lazy loading way 3 JavaScript code	114
93.	Well-known URL identifiers variations and patterns	115
94.	Type matching to discover additional interfaces which expose an attribute whose typing is an identified interface	116
95.	History interface.	117
96.	Browser history manipulation using the History and Location interfaces before manipulation.	117
97.	Browser history manipulation using the History and Location interfaces after manipulation	119
98.	Browser history manipulation using the History and Location interfaces live web	120
99.	Object vs Function Object Web IDL	128
100.	Object vs. Function Object JavaScript	129

101. HTMLAudioElement named constructor Web IDL	129
102. HTMLAudioElement named constructor JavaScript	130
103. Interface attribute and operation patch overrides	130
104. Prototype object patch modification	131
105. Interface existing instance replace modification	131
106. Global execution object replace modification	132
107. Existing instance replace plus patch	132
108. Existing instance replace plus patch modification	133
109. Foreign substitution unforgeable Location interface	134
110. Archive-It rewriting the text string “location” in the archived JavaScript of mendeley.com user pages. Live web version on the right.	134
111. Pywb version 0.33 rewriting the text string “location” found in non- JavaScript page markup for the documentation of React Router	135
112. Text string “location” in HTML bundled with the JavaScript for the documentation of React Router	136
113. Incorrect rewriting of the text string “location” and “top” in React Router’s documentation. The incorrect rewrites are highlighted in red (Figure 113a) and the original strings in the documentation are highlighted in green (Figure 113b)	138
114. Webrecorder and Pywb global patch override for its rewriting of the text strings “location” and “top”	138
115. Archive-It’s mitigation for its rewriting of the text string “location” and “top”	139
116. Example extend override	140
117. Archive Window and Document interface proxies	141
118. Archive JavaScript proxy setup anonymous block scope	142
119. Archive JavaScript proxy setup function scope	143
120. Browser compatibility tables for the let declarator and JavaScript Proxy object	144

121. Location checks negated by archive controlled window proxy	145
122. Temporal spread of the composite mementos crawled	149
123. Alexa June 2017 rankings of the composite mementos crawled	150
124. Cumulative number of requests (Figure 124a) and number of requests per page (Figure 124b) for 577 composite mementos replayed from the Internet Archive's Wayback Machine.	154
126. Cumulative number of client-side rewrites (Figure 125a) and number of client-side rewrites client-side per page (Figure 126a) for 577 composite mementos replayed from the Internet Archive's Wayback Machine.	156
127. Cumulative number of blocked requests (Figure 127a) and number of blocked requests per page (Figure 127b) with and without client-side rewriting for 577 composite mementos replayed from the Internet Archive's Wayback Machine	160
128. ΔReq and $\Delta Req'$ values for 577 composite mementos replayed from the Internet Archive's Wayback Machine	162
129. Instagram replayed from the Internet Archive	163
130. Instagram archived location reloading JavaScript	164
131. Archived page that is to just set a cookie and reload the page	164
132. m.vk.com cookie setting and location replacing JavaScript	165
133. The home page of cnn.com is replayable from the Internet Archives Wayback Machine with client-side rewriting	167
134. The home page of reuters.com increased replay fidelity from the Internet Archive's Wayback Machine with client-side rewriting	169
135. Home page of sohu.com increased replay fidelity from the Internet Archive's Wayback Machine with client-side rewriting	172
136. soufeel.com increased replay fidelity from the Internet Archive's Wayback Machine with client-side rewriting,	174
137. Malicious XMLHttpRequest targeting the banner of the Internet Archive's Wayback Machine	175
138. Embedded JavaScript delivering EvilXHR executed before Wayback Machine's toolbar.js	176

139. Timemap retrieved from the Wayback Machine for the page with malicious XMLHttpRequest	177
140. Wayback Machine banner code, toolbar.js, affected by EvilXHR	177
141. Evil <code>createElement</code> targeting the display of the memento selection feature of the Wayback Machine's banner,	178
142. Red selection bar for choosing a different memento to view	178
143. Embedded JavaScript delivering evil <code>createElement</code> executed before Wayback Machine's toolbar.js	179
144. Wayback Machine banner code, toolbar.js, affected by evil <code>createElement</code>	180
145. <code>http://wayback.archive-it.org/wb-static/js/ait-client-rewrite.js</code>	202
146. Full Response containing the portion of un-rewritten HTML shown in Figure 83	203

CHAPTER 1

INTRODUCTION

“When you have eliminated the JavaScript, whatever remains must be an empty page”¹

“Time keeps on slippin’, slippin’, slippin’ into the future . . . I want to fly like an eagle, to the sea” is an excerpt taken from the refrain and chorus of *Fly Like An Eagle* by the Steve Miller Band. These lyrics from the song speak about how quickly the present becomes the future, and to understand the passing of time one must lift himself to a higher plain like the eagle who flies in sky towards the sea. This is much like web archiving. Web archiving (eagle) preserves the web (sea) over time so that our future generations can know the past in order to understand the future. But unlike the traditional sea, the digital ocean that is the web changes at a much faster rate over time.

A four-year longitudinal study that was conducted from December 1996 to February 2001 of 361 web pages measuring their change in both content and “liveliness” on a weekly basis [1] found that 34.4% of the original sample set did not exist by the end of that study, while another study in 2001 [2] estimated that only 20% of “today’s” web pages will be “alive” after a year. An earlier study in 1997 [3] found that out of 950,000 AT&T web trace records, 16.5% of the sample set was changed at each access and the primary modifications were to link and image tags. The issue of changing and missing web content is such an issue that a group of researchers created a corpus of web pages containing dead links entitled “Book of the Dead” [4, 5, 6]. This was the motivating factor for the start of web archiving in 1996 when Brewster Kahle founded the Internet Archive², a non-profit digital library with the goal of preserving the entire ethos of the web. The goal was not only to simply preserve the web’s content, but to relive it through the Wayback Machine(s)³, which allows its users to replay archived web pages at any given point of time.

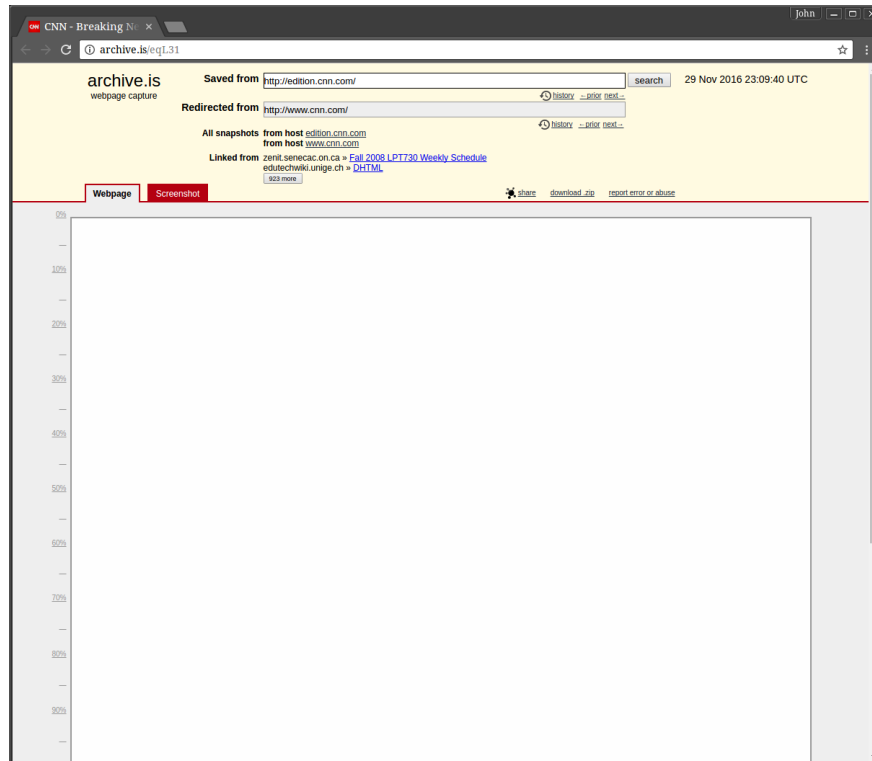
¹From the *noscript* tag of a Google web page

²<https://archive.org/about/bios.php>

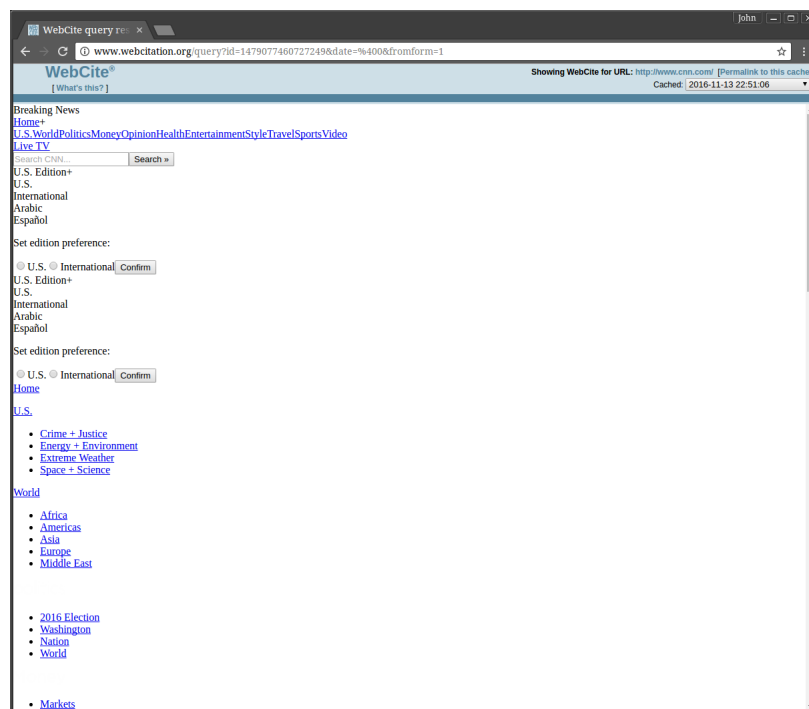
³http://web.archive.org/web/20020806031346/http://www.archive.org:80/wayback/press_kit/press_release.html

But as time has progressed, more and more web users desire to preserve and replay web content that was not available at the inception of the modern web archiving infrastructure [7, 8, 9, 10]. Take, for instance, the issue of “unarchivable” web pages with the disappearance of a popular US news network CNN, and, its main web page from the primary web archives right before the presidential election of 2017 [11]. Figure 1 shows the archived versions of *www.cnn.com* replayed from three of the more widely known archives, archive.is (Figure 1a), WebCitation (Figure 1b), and the Internet Archive (Figure 1c).

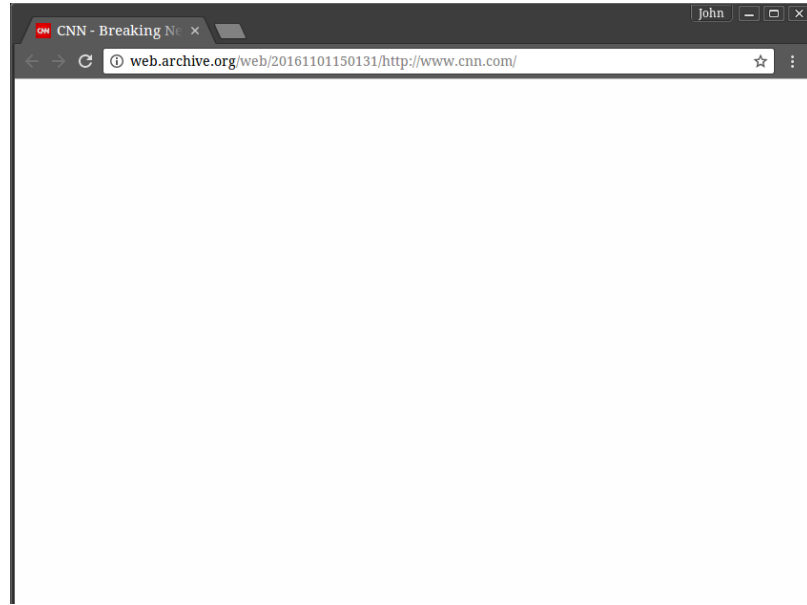
The primary issue is with the changes CNN made to their content delivery network (CDN) and replaying the page’s JavaScript, which retrieves the page’s resources from the CDN. The live web page uses an older technique in respect to web development standards to overcome the restrictions placed on web pages by Same Origin Policies by setting the domain of the document to a shorter common domain. But when the page is replayed from the archive, the Same Origin Policy [12] disallows changing the domain because the origin is now the archives (e.g., *web.archive.org*), not *cnn.com*, causing the information that the page’s JavaScript required to load the page’s resources to “disappear” due to browsers enforcement of the Same Origin Policy. Though the archival failure for this page exists across multiple archives, the page also demonstrates additional archive specific issues; when replayed from WebCitation (Figure 1b), the page appears to be “properly archived”, albeit without any CSS styling.



(a) Replayed from archive.is on 2016-11-29



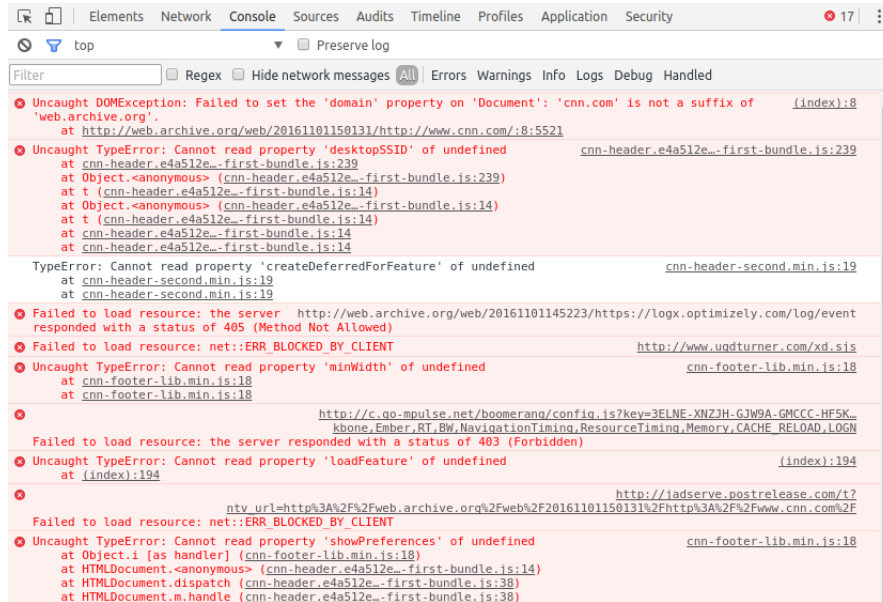
(b) Replayed from WebCitation on 2016-11-13



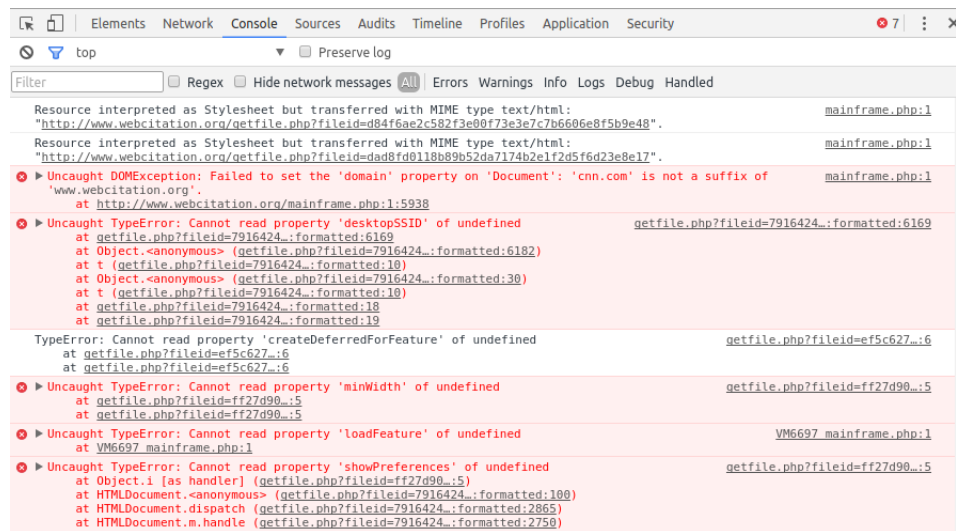
(c) Replayed from the Internet Archive on 2016-11-01

Fig. 1. Replay failure of the home page for CNN.com

The difference in replay from the Internet Archive (Figure 2a) and WebCitation (Figure 3a) is that WebCitation changed the *Multipurpose Internet Mail Extensions* (MIME) [13, 14] type of the pages style sheets from “text/css” to “text/html” which in turn caused the browser to not apply the archived *Cascading Style Sheets* (CSS) to the page; hence, the appearance of non-failure in preservation. If WebCitation had sent the correct MIME type there would be little apparent difference from replay on the Internet Archive (Figure 2a), as both execute the page’s JavaScript and both incur the Same Origin Policy issue.



(a) Errors when the home page of CNN is replayed from the Internet Archive



(a) Errors when the home page of CNN is replayed from WebCitation

Fig. 3. Replay errors for the home page of CNN as seen by the developer console

Conversely, archive.is does not execute the archived page's JavaScript at replay time nor does archive.is preserve the page's JavaScript. Like Webcitation, archive.is modifies the page but in its entirety by applying a transformation to the original markup. Figure 4 shows the page, with the "white out" removed, replayed from archive.is with the elements page of the Chrome developer tools open. The elements

page shows us that archive.is has applied a transformation to the page’s original HTML in order to preserve and replay it using the archive’s closed source replay engine.

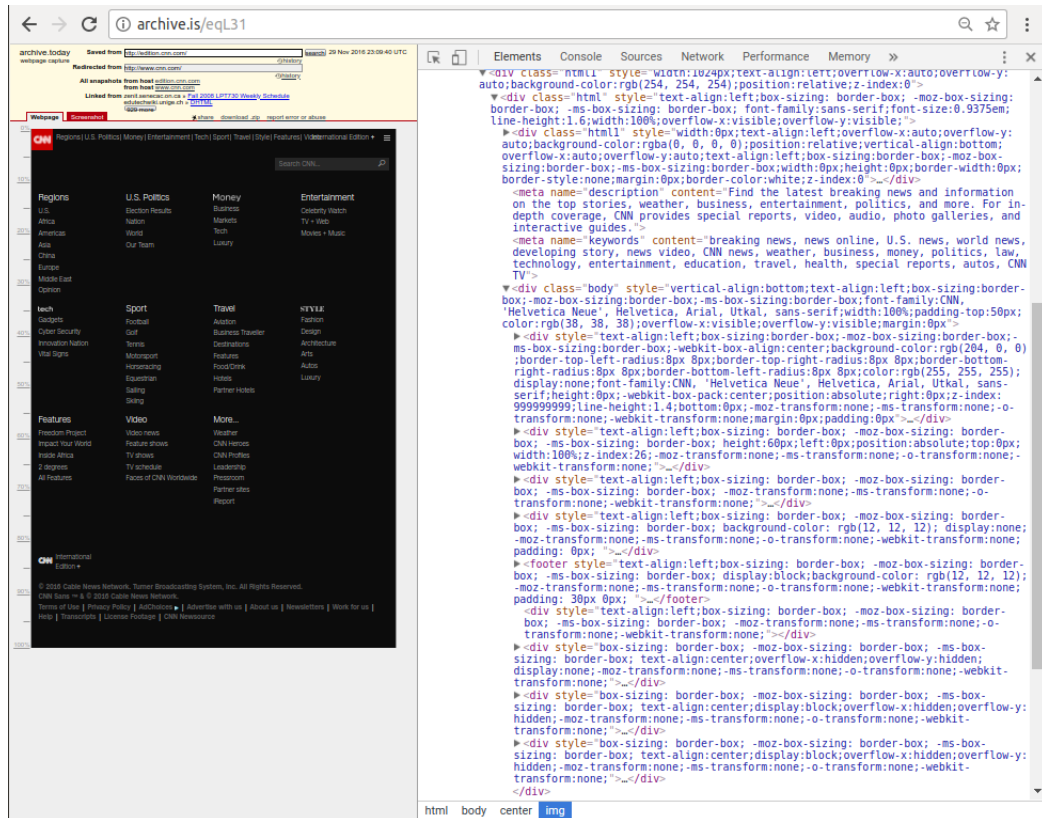


Fig. 4. archive.is archival transformation of CNN

This allows archive.is to remove the original representation’s HTML attributes, inline the style sheet definitions that the browser would normally apply to the elements, and create the foreign representation of the page shown in Figure 4. This style of modification is primarily made by archival services that do not use the “Wayback” model of replay. A “Non-Wayback” archival service is one that may not follow the standard model of web archiving of preserving the original representation and making it accessible in both original and rewritten formats via an implementation of the Internet Archive’s Wayback machine (Chapter 4). Ultimately, modifications rely on the “good intentions” on the part of the archiving entities due to the lack of vocabulary, guidelines, or preservation framework by which to ensure these modifications are not completely detrimental to both replay and preservation of the web.

An example of the negative effects of an archive’s direct modifications of page can be


found when replaying a seemingly non-complicated web page archived by the Archive-It service. Figure 5a shows <https://www.mendeley.com/profiles/helen-palmer/>, which will always redirect [15] when attempting to replay the page from Archive-It (Figure 5b) and any archive that preserves and execute the page's JavaScript (Figures 5c and 5d)⁴.

The screenshot shows the Mendeley profile page for Helen Palmer. The browser address bar displays <https://www.mendeley.com/profiles/helen-palmer/>. The page header includes the Mendeley logo, a search bar, and links for 'Create a free account' and 'Sign In'. The profile section features a circular profile picture of Helen Palmer, her name 'Helen Palmer', and her credentials: 'PhD', 'Post doctoral research fellow', and 'Department of circulation and medical imaging NTNU'. It also shows '0 Readers' and '4 Publications', along with a 'Follow' button. Below the profile, there are tabs for 'Overview' and 'Network'. The 'Overview' tab is active, showing 'Research interests' (Neuroscience: Exercise and cognition), an 'About' section (Educated as a physiologist. Currently working with neuroimaging. Developing research projects in the field of exercise and brain function.), and 'Followers (3)' (Alexander Olsen and Martin Wohlwend). The 'Publications' section lists two papers: 'Exercise training for a time-poor generation: enhanced skeletal muscle mitochondrial biogenesis.' (13 Readers) and 'Optogenetic fMRI sheds light on the neural basis of the BOLD signal.' (82 Readers).

(a) <https://www.mendeley.com/profiles/helen-palmer/> as seen on the live web

⁴<https://youtu.be/GzfMOEgaB24>

wayback.archive-it.org/8130/20161215231900/http://www.mendeley.com/sign-in/?routeTo=https%3A%2F%2Fwww



SMJ: Tool Archivability Study (Second Pass) Web Archive (Old Dominion University)

Enter Web Address: All

Not in Archive

The page you requested has not been archived in Archive-It.

This could be for a number of reasons.

Most likely the page you are requesting was outside of the crawler's scope. Try another request or click to see other pages from www.mendeley.com.


It is also possible that this site is currently being crawled and the archived pages are not yet available in the Wayback Machine. It can take up to 24 hours after a crawl has been completed for the site to appear in the Wayback Machine. Please try again after the allotted time.

You can also try searching for www.mendeley.com/sign-in/?routeTo=https%3A%2F%2Fwww.mendeley.com%2Fprofiles%2Fhelen-palmer on the [live web](#), in the [global Wayback Service](#), or in the General Archive at the Internet Archive at www.archive.org

[Home](#) | [Internet Archive](#)

(b) <https://www.mendeley.com/profiles/helen-palmer> replayed from Archive-It

web.archive.org/web/20170126164831/https://www.mendeley.com/sign-in/?routeTo=http%3A%2F%2Fweb.archiv



<https://www.mendeley.com/sign-in/?routeTo=http%3A%2F%2Fweb.archiv>

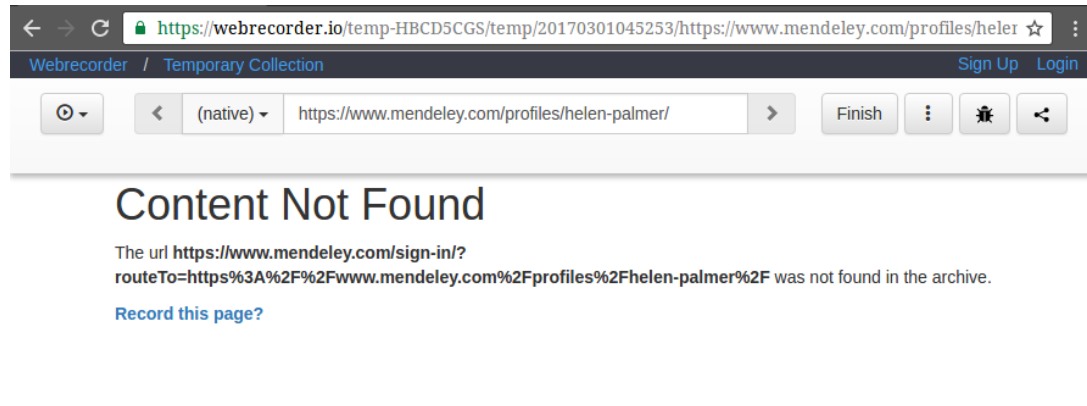
Latest

Show All

Hrm.

Wayback Machine doesn't have that page archived.

(c) <https://www.mendeley.com/profiles/helen-palmer> replayed from The Internet Archive



(d) `https://www.mendeley.com/profiles/helen-palmer` replayed from Webrecorder

Fig. 5. Replay failures of `https://www.mendeley.com/profiles/helen-palmer`

When inspecting the archived versions of the page's embedded resources when replayed by Archive-It and Webrecorder you would discover that the pages' JavaScript varies drastically from the original. Both Archive-It and Webrecorder are re-writing well-known global JavaScript identifiers with their own overrides (Figure 6).

```

1 // archive it version
2 window.__PRELOADED_STATE__ = {
3   "WB_wombat_self_location": {
4     "id": "fcc2fd44-d82e-45ca-8855-35ee6b8bfbe9",
5     "latitude": 63.44,
6     "longitude": 10.4,
7     "name": "Trondheim, Norway",
8     "city": "Trondheim",
9     "state": "Sør-Trøndelag",
10    "country": "Norway"
11  },
12 };
13
14 o.Auth.authCodeFlow({
15   authenticateOnStart: !1,
16   apiAuthenticateUrl: function() {
17     var t = "/sign-in/?routeTo="
18     encodeURIComponent(WB_wombat_self_location);
19     return WB_wombat_self_location = t
20   },
21   refreshAccessTokenUrl:
22     ↪ "/profiles/refreshToken/"
23 });
24
25 function s(t) {
26   var e = t.headers.WB_wombat_self_location;
27   if (e && this.settings.followLocation &&
28     201 === t.status) {
29     {method: "GET",url: e,responseType:
30       ↪ "json"};
31     return this.send(n);
32   }
33   return t.headers.link && "string" == typeof
34     ↪ t.headers.link
35     && (t.headers.link = l(t.headers.link)), t;
36 }

```

Actual location

```

1 // live web version
2 window.__PRELOADED_STATE__ = {
3   "location": {
4     "id": "fcc2fd44-d82e-45ca-8855-35ee6b8bfbe9",
5     "latitude": 63.44,
6     "longitude": 10.4,
7     "name": "Trondheim, Norway",
8     "city": "Trondheim",
9     "state": "Sør-Trøndelag",
10    "country": "Norway"
11  },
12 };
13
14 o.Auth.authCodeFlow({
15   authenticateOnStart: !1,
16   apiAuthenticateUrl: function() {
17     var t = "/sign-in/?routeTo=" +
18     encodeURIComponent(location);
19     return location = t
20   },
21   refreshAccessTokenUrl:
22     ↪ "/profiles/refreshToken/"
23 });
24
25 function s(t) {
26   var e = t.headers.location;
27   if (e && this.settings.followLocation &&
28     201 === t.status) {
29     {method: "GET",url: e,responseType:
30       ↪ "json"};
31     return this.send(n);
32   }
33   return t.headers.link && "string" == typeof
34     ↪ t.headers.link
35     && (t.headers.link = l(t.headers.link)), t;
36 }

```

Not actual location

Not actual location

Fig. 6. Archive-It rewriting the identifier “Location” in the archived JavaScript of mendeley.com user pages

This technique of sever-side JavaScript rewriting is useful for ensuring that archived JavaScript cannot initiate browser navigation to the live web when using the “location” identifier found on the global window object [12] and is used in combination with client-side rewriting. Because Archive-It’s and Webrecorder’s server-side rewriting were targeting all instances of the “location” identifier even if they were not responsible for browser navigation, their client-side rewriting libraries had to make modifications to the archived JavaScript and the JavaScript environment of the browser. Depending on the replay system’s rewrite mechanisms, the JavaScript of the page could collide with the replay system, causing undesired effects. But that was not the true reason of the continual redirection; rather, there is no web archiving standard on the replay of HTTP Cookies which the page required for authentication even on pages accessible without an account (Chapter 5).

Thus, the page’s JavaScript thought the replayed page being viewed required the viewer to sign in and will always cause redirection to happen before the page has

loaded. It is these issues that exemplify a host of other issues that are challenging the web archiving community's perception of performance [16] and capture [17].

1.1 CONTRIBUTIONS

The web archiving community lacks the terminology to describe the existing styles of replay, which are dependent on the policies of the web archive, and the modifications made to an archived web page and its embedded resources in order to facilitate replay. Because of this, there are only ad hoc standards for the replay of archived web pages. To provide the terminology necessary for describing both the existing styles of replay and the modifications made to facilitate replay, which will hopefully improve the replay of archived web pages, the contributions of this thesis are as follows:

- Provide a classification of and terminology for the current styles of replay (Chapter 4)
- Provide a classification of and terminology for the modifications made to an archived web page to facilitate replay (Chapter 4)
- Propose a standard and generalized method for the generation of client-side rewriting libraries (Chapter 4)
- Provide a classification of and terminology for the modifications made to an archived web page's JavaScript in order to facilitate client-side rewriting (Chapter 5)
- Detail a combination server-side and client-side rewriting technique that decreases the amount of modifications made to archived JavaScript and provides an archive more control over replay (Chapter 5)
- Evaluate the effectiveness that client-side rewriting would have in augmenting already existing server-side rewriting systems of an archive (Chapter 5)

1.2 THESIS ORGANIZATION

This thesis has six chapters. In Chapter 2, we discuss the concepts necessary to understand replay of archived web pages. We provide an overview of HTTP, HTML, JavaScript, how viewing a web page in a web browser works, Memento, the archival and replay of web pages, and URL rewriting. In Chapter 3, we discuss previous works that have also looked at the issues surrounding the archival and replay of web pages.

In Chapter 4, we progressively classify and define terminology for the existing styles of replay and the modifications made by them to facilitate replay by considering in detail how the Internet Archive, Webrecorder and archive.is replay the contents of their archives. In Chapter 5, we propose a method for the auto-generation of client-side rewriting libraries through the usage of the Web Interface Design Language [18] definitions for the JavaScript APIs of the browser. We also provide classification and terminology for the modifications made to the JavaScript environment by the client-side rewriter and provide a solution to decrease the amount of server-side rewriting performed on archived JavaScript in order to, for example, ensure JavaScript can not navigate the browser to the live web (Figure 6). Finally, in Chapter 5, we evaluate the effectiveness of the generated client-side rewriting library in augmenting the already existing server-side rewriting systems of the Internet Archive’s Wayback Machine. In Chapter 6, we discuss the conclusions from this work, the contributions of this work and future work, which could be done as an extension of this thesis.

CHAPTER 2

BACKGROUND

In this chapter, we will discuss the concepts necessary to understand the remainder of this thesis. We will discuss the Hyper Text Transport Protocol (HTTP), the primary means by which the information on the web is sent and received. Then we will discuss Hyper Text Markup Language (HTML), the primary markup language for creating pages and applications on the web. Then we will discuss JavaScript, the programming language of the web and will give an overview of the rules involved when viewing a web page using a web browser. Then we will discuss the Memento protocol, an extension to HTTP to support versioning and archiving. Finally, we will discuss archiving and replaying pages, closing with a discussion on URL rewriting.

2.1 HYPER TEXT TRANSPORT PROTOCOL (HTTP)

HTTP [19] is the primary protocol for retrieving or modifying resources [20] on the World Wide Web (Web). Each web accessible resource is identified by a **Uniform Resource Identifier (URI)** [21], or more commonly known as Uniform Resource Locator (URL). The primary distinction between an URI and URL is that an URL “refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism”, whereas a URI may be used to identify or name any conceivable resource [21].

As shown in Figure 7, the first part of a URI is the scheme. A URI’s scheme defines the **protocol** used during **dereferencing** (a process by which you retrieve the resource’s representation). The scheme for a URI does not necessarily have to be either HTTP [19] or HTTPS [22] (HTTP over Transport Layer Security) but may be protocol-less or some other defined protocol such as ftp. After the URI scheme comes its authority which is comprised of host and optional port. The authority for an URI states the host (owner) of the resource and is also used by the browser to apply security polices on a resource (which will be discussed further on in this chapter). The optional port portion of the authority is used for the TCP portion of HTTP(S) and is beyond the scope of this thesis. Following a URI’s authority

is the path. The path for a URI “serves to identify a resource within the scope of the URI’s scheme and naming authority (if any)” [21]. The query portion of a URI represents non-hierarchical data if any and is used to execute a query against the resource the URI identifies. Note that the only hierarchical portions of a URI are the authority and path. The final part of a URI is the fragment, which identifies a secondary resource contained in the resources itself.

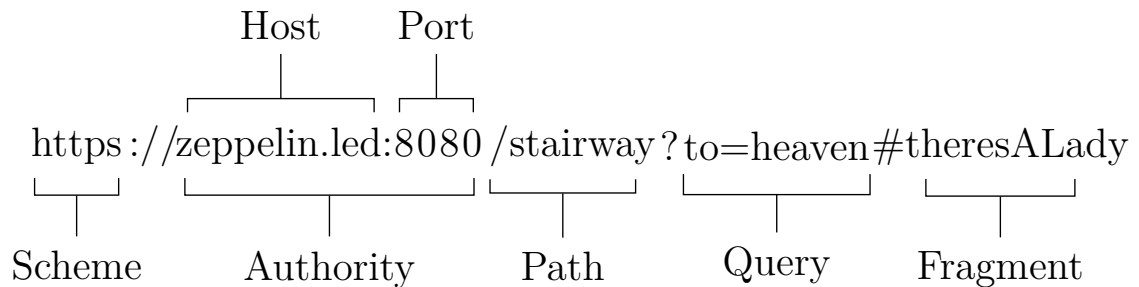


Fig. 7. Anatomy of a URI

All of this information is used by a user-agent during the process of retrieving the representation of a URI, called dereferencing, in the form of **request** and **response** pairs. A HTTP request is made to the origin server by a client using one of the available HTTP methods [23]. Table 1 lists a subset of the available methods. Each HTTP method has its own semantics and usage. For example, a **GET** request is used to ask the origin server for the representation of a URI, and an **OPTIONS** request is used to ask the origin server for specific details about the available methods for the target URI.

Table 1
RFC 7231 HTTP methods

Method	Usage
GET	Request to receive a resources representation.
HEAD	Request only the HTTP headers of the GET request.
POST	Request the resource to process the body of request.
PUT	Request to create or update the resource using the request's body.
DELETE	Request to delete or remove the resource.
OPTIONS	Request for information concerning communication options for the resource

The requests and responses follow a general format which can be seen in Figure 8. A request will begin with a *request line* which is comprised of the HTTP method, the path of the URI and the HTTP version used. Following the *request line* is a series of HTTP headers. HTTP headers [23, 24] convey additional information about the request or response and are also a required part of *content negotiation* (Figure 8).

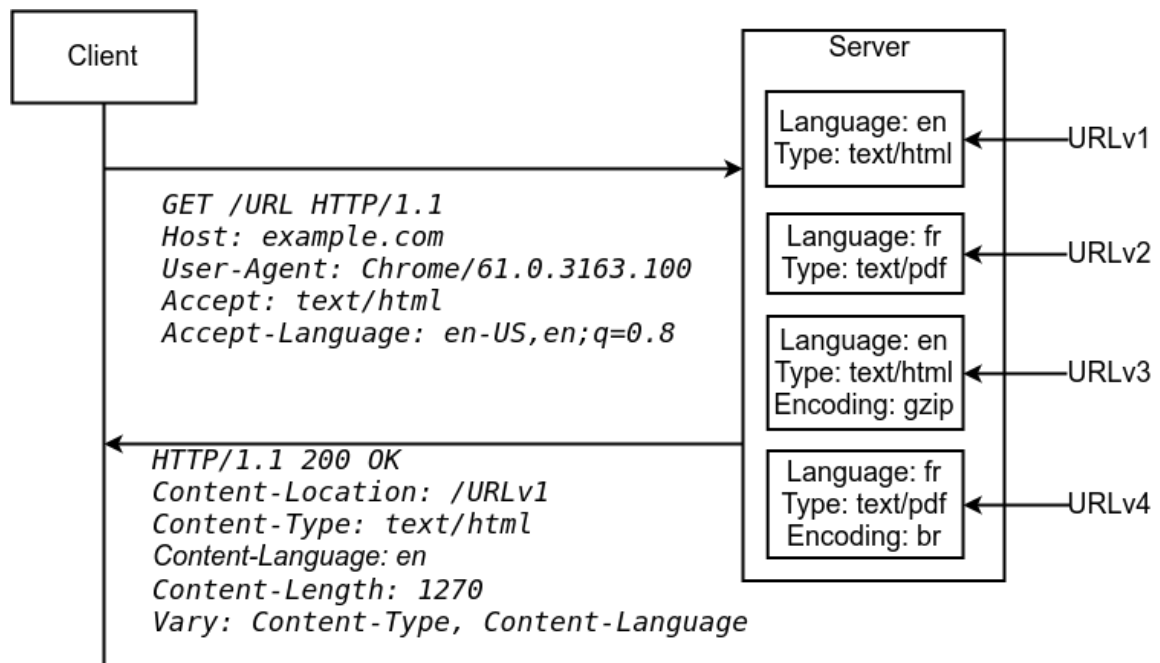


Fig. 8. Anatomy of content negotiation

The content negotiation process is started when a client de-references a URI by sending HTTP headers indicating the desired representation of the URI (*Accept-Type*) and the preferred language (*Accept-Language*). When the server receives the request, the server determines if it has the corresponding representation indicated by the client. The server then replies with a response that begins with a response line comprised of the HTTP version, the status code and reason followed by HTTP Header fields that indicate which representation was sent (Figure 8). The *Content-Location* header indicates to the client the actual URI of the representation if there are more than one, the *Content-Type* header indicates the type of the representation sent, and the *Content-Language* header indicates which language the representation is in. The *Vary* header informs the client in which dimension content negotiation was applied. The size of the response body, if there was one, sent by the server is indicated by the *Content-Length* header. The status code plus reason indicates if the request was successful or not. Table 2 lists the status code types alongside their corresponding meaning.

Table 2
HTTP status code classes

Status	Class	Meaning
1xx	Informational	The server understood the original request and the client may proceed
2xx	Successful	The request completed and there maybe a response
3xx	Redirection	The first URI has moved to a new URI (location)
4xx	Client Error	Your request was faulty
5xx	Server Error	The server ran into an issue

For example, if the request was successful (Figure 8) then the response code and reason would be 200 OK. If the server wished to indicate that the resource requested using the initial URI has moved to another location, it would use the status code 302 and including the location header in the response's HTTP Headers. An example of this can be seen in Figure 9, which highlights the content negotiation process for redirection.

As seen in Figure 9, when the client receives the response for the initial request, the location header field indicates that the representation for the URI `http://example.com/example` is temporarily located at a new URI `http://example.com/overhereNow`. Now the client makes another request dereferencing the new URI and then received a response with status 200 OK and ending negotiations. The final portion of the response is the response body which does not necessarily have to be included as is the case for a 302 response. Additional cases for when a response may not contain a response body is when the status code for the response is 204 No Content or 404 Not Found, indicating there is no content or the request's URI does not exist on the server, respectively.

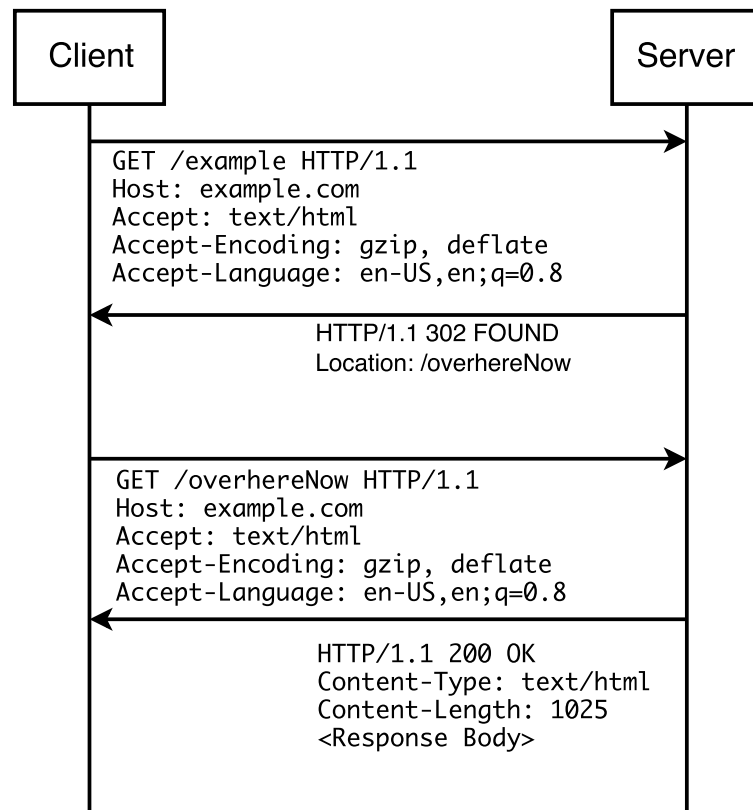


Fig. 9. HTTP request with redirection

2.2 HYPER TEXT MARKUP LANGUAGE

Hyper Text Markup Language (HTML) is the most popular document type for representing web pages. 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 elements. The first of these tags is the `html` tag. Figure 10 contains the HTML for the web page `http://example.com` and will be the listing used when referring to line numbers, e.g. tag line 1. A rendering of the HTML shown in Figure 10 when rendered using the Google Chrome web browser can be seen in Figure 11.

The `html` tag on line 2 is considered the document element and is the root of HTML documents. All other elements in the document are considered a child of this tag. The first child of the `html` tag is the `head` tag and contains elements that define metadata about the document or bring in or define resources for the page. The contents of the `head` tag of Figure 10 (lines 4 through 5) contain a single `title` tag, three `meta` tags, and a single `style` tag. The `title` tag defines the title of the page and the `style` tag contains the CSS style definitions for the page. One may also include CSS by using the `link` tag. There are two `meta` tags which demonstrate the ability for the page's creator to control the inner workings of the browser or apply additional HTTP headers through this tag. Figure 10 (lines 6 and 7) contains two `meta` tags that provide that level of control for the pages' creator.

```

1  <!doctype html>
2  <html>
3  <head>
4      <title>Example Domain</title>
5      <meta charset="utf-8" />
6      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
7      <meta name="viewport" content="width=device-width, initial-scale=1" />
8      <style type="text/css">
9          body { background-color: #f0f0f2; margin: 0; padding: 0;
10             font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
11             }
12         div {
13             width: 600px; margin: 5em auto; padding: 50px;
14             background-color: #fff; border-radius: 1em;
15             }
16         a:link, a:visited { color: #38488f; text-decoration: none; }
17         @media (max-width: 700px) {
18             body { background-color: #fff; }
19             div { width: auto; margin: 0 auto; border-radius: 0; padding: 1em; }
20         }
21     </style>
22 </head>
23 <body>
24 <div>
25     <h1>Example Domain</h1>
26     <p>This domain is established to be used for illustrative examples in documents.
27     You may use this domain in examples without prior coordination or asking for
28     permission.</p>
29     <p><a href="http://www.iana.org/domains/example">More information...</a></p>
30 </div>
31 </body>
32 </html>

```

Fig. 10. The HTML of the famous `http://example.com`

The first `meta` tag in Figure 10 contains the attribute *http-equiv*¹. The *http-equiv* attribute for a `meta` indicates that the value of the tag's `content` attribute is used to simulate the exact same behavior as if it were included in the HTTP headers of the response for the page. HTTP headers may be used in a `meta` tag if and only if the value for *http-equiv* is a valid HTTP Header and is one of the allowed HTTP headers allowed for this tag, namely **Content-Type**, **Content-Language**, **Content-Security-Policy**, and **Set-Cookie**. The second `meta` defines how mobile browsers should render the page. The only tag not found in the `head` tag of Figure 10 that is typically included is a `script` tag which contains or brings in JavaScript.

After the `head` tag, is the `body` tag which contains the content of the document and may include every HTML element defined in the HTML specification [12]. In addition to the attributes defined for an element, the fifth revision of the HTML

¹Note that this thesis uses the convention of denoting HTML tags, JavaScript objects or functions and like entities using typewriter text styling whereas we use italics to denote their attributes

specification allows for custom attributes to be on any given element [25], not just those defined for the element.

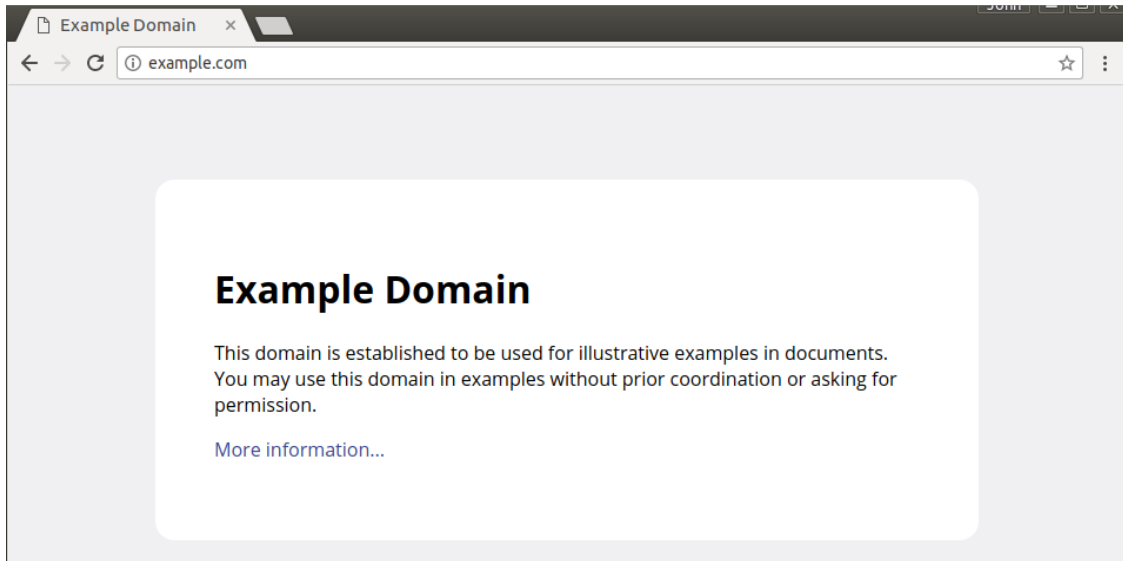


Fig. 11. `http://example.com` rendered by Google Chrome

2.3 JAVASCRIPT (JS)

JavaScript (ECMAScript) is the programming language of the web [26]. Even though the full capabilities of the JavaScript programming language is beyond the scope of this thesis, the ability for JavaScript to make additional HTTP requests, create and manipulate the HTML document (DOM), and, to some extent, control the location of the browser are within this thesis's scope. An example for each of JavaScript's capabilities that will be discussed can be seen in Figures 12 and 16. Example 1 shows us four things; the first is making a complex request (Figure 12, lines 1-9).

```

1 // Example 1: make request for an image
2 // and include the cookie sent by the server in the request headers
3 async function getImg (imageURL) {
4   let blob = await fetch(imageURL, { credentials: 'include' }).then(res => res.blob())
5   const body = document.querySelector('body')
6   const myImage = new Image()
7   myImage.src = window.URL.createObjectURL(blob)
8   body.appendChild(myImage)
9 }

```

Fig. 12. JavaScript making an HTTP request for an image and mutating the DOM by adding an `img` tag whose `src` attribute is a Blob URL

The function `getImg` makes a request for an image using “credentials” (line 4). When the request is sent, it includes the authentication information (cookies) the server sent with the response for the root page this JavaScript snippet would be associated with. The typical use case for using the credentials sent over by the server in a JavaScript initiated request is to validate that the user has permission to view the resource. We will not go into details about HTTP authentication [27, 28] as it is out of scope for this thesis. Once the response for this request comes back, the raw data of the image is transformed into a **Blob URL** [29] and a new HTML Image is added dynamically to the original document using the JavaScript DOM APIs (lines 5-8) [30, 12].

```

11 // Example 2: dynamically adding scripts
12 const body = document.querySelector('body')
13 const script1 = document.createElement('script')
14 script1.src = scriptURL
15 body.appendChild(script1)
16 const script2 = document.createElement('script')
17 script2.innerHTML = jsString
18 body.appendChild(script2)
19 eval(anotherStringOfJs)

```

Fig. 13. Ways additional JavaScript can be introduced into the page

Example 2 (Figure 13) demonstrates three different ways additional JavaScript can be introduced into the page. The first way additional JavaScript can be introduced into page (lines 12-15) is similar to the first example in that the newly created `script`

tag has its *src* attribute set to URL that points to a JavaScript file. Setting the *src* attribute to a valid URL pointing to a JavaScript file causes the browser to initiate an additional resource fetch (HTTP GET request) [12], and when the response comes back, the browser executes the contents of that file. The second way additional JavaScript can be introduced into the page (lines 16-18) also creates a new `script` tag, but instead of requiring an additional HTTP request for the contents of the script tag, the *innerText* attribute of the tag may be set to a valid string of JavaScript code. When this tag is added to the document, the browser will execute the JavaScript code inside the tag. The third way, (Figure 13 line 19), demonstrates how JavaScript does not need to use a script tag to introduce additional JavaScript code into the page but can use the `eval` function [31]. The `eval` function will execute any string of valid JavaScript code that is supplied to the function; this string may come from any source as long as it is indeed valid JavaScript.

```
// Example 3: set cookie for the browser and document
document.cookie = 'partyOnWayne=partyOnGarth'
```

Fig. 14. JavaScript adding an HTTP cookie to the page

Example 3 (Figure 14) shows how JavaScript can add additional cookies to the browser. If, for instance, example 3 was executed before example 1 (Figure 12, lines 3-9) then the additional cookie added to the browser would be sent alongside any additional cookies the browser had before example 3 was added. Example 4 (Figure 15) demonstrates how JavaScript can control the location of the browser. When `window.location` is set to any other valid URL other than the current location, it will cause the browser to navigate away from the currently viewed page.

```
// Example 4: set the windows location
window.location = 'http://someWhereElse.com'
```

Fig. 15. JavaScript changing the location of the browser

Example 5 (Figure 16, lines 3-5), demonstrates how JavaScript can manipulate the history of the browser and change both the *Referrer* HTTP header sent by the browser and the page's title [12]. The `pushState` function of the history object (line 3) takes three arguments, the first is history state (information about the “new location”), the

second argument is the new title for the page, and the third argument is the “new location” (displayed in the location bar of the browser). Only the `pushState` function can change the *Referrer* HTTP header and add a new history entry to the browser. Whereas the `replaceState` function (line 5), which takes the same arguments as `pushState`, only replaces the current history entry while still updating the page’s title and location displayed in the navigation bar of the browser.

```
1 // Example 5: manipulate the history of the browser without navigation
2 // changes the referrer HTTP header sent by browser
3 history.pushState({info: 'xyz'}, 'Awesome Page', '/page.html')
4 // modify current history entry
5 history.replaceState({info: 'abc'}, 'Another Page', '/page2.html')
```

Fig. 16. JavaScript for changing browser history

Example 6 (Figure 17) demonstrates how JavaScript can change the domain of the page, if and only if the new domain is parent domain of the current origin [12].

```
// Example 6: change the domain of the pages only valid if the
// new domain is a parent domain of the original
document.domain = 'abc.com'
```

Fig. 17. JavaScript for changing the domain of the browser

2.4 BROWSERS AND VIEWING A WEBPAGE

As previously discussed in Section 2.1, the resources on the web are identified by URIs; to retrieve the representation (content) of these URIs, you must use a process called dereferencing. This is what the browser does when it navigates to the URI you enter into the navigation bar of the browser. The browser will parse the HTML and dereference additional URIs contained in the HTML elements for images or other embedded resources. URIs associated with script tags are fetched and executed while the script tags with in-line JavaScript are executed. The style sheets for a page are fetched and parsed, bringing in additional styling resources such as fonts or images alongside embedded images, video, or audio files, which are also fetched and applied to the document.

All this happens when you first view the page and is not limited to dynamic additions of HTML elements or HTTP requests by JavaScript. A noteworthy and

important HTML element that can bring in an entirely new browsing context (full HTML page at another URI) is the carrier `iframe` element. This element, when introduced into the page, behaves as if you had navigated the browser to its URI, but security restrictions limit its interactivity with the embedding page. It is these security restrictions on which the remaining portions of this section will focus; namely, the origin based restrictions placed on resource fetches by JavaScript. The browser's (document's) origin [32, 12] is set when the browser navigates to the URI. For example, when your browser navigates to `http://awesome.example.com`, the origin of the document is set to `awesome.example.com`.

This means that if the page at `awesome.example.com` wished to use its embedded JavaScript to make requests to or load additional resources from `super.example.com`, these requests would be restricted by the Cross Origin Resource Sharing (CORS) protocol [33]. CORS can be explained concisely in the flowchart in Figure 18, which states that anything other than a simple request to the remote server requires the browser to retrieve a server's authorization for the request. The term "simple" refers to the HTTP methods (GET, HEAD, and POST) which make requests with the *Content-Type* HTTP header set to *application/x-www-form-urlencoded*, *multipart/form-data*, or *text/plain*. Any other method or request for content type would make GET, HEAD or POST requests non-simple. This also includes the usage of custom headers, even on request methods that would be considered simple.

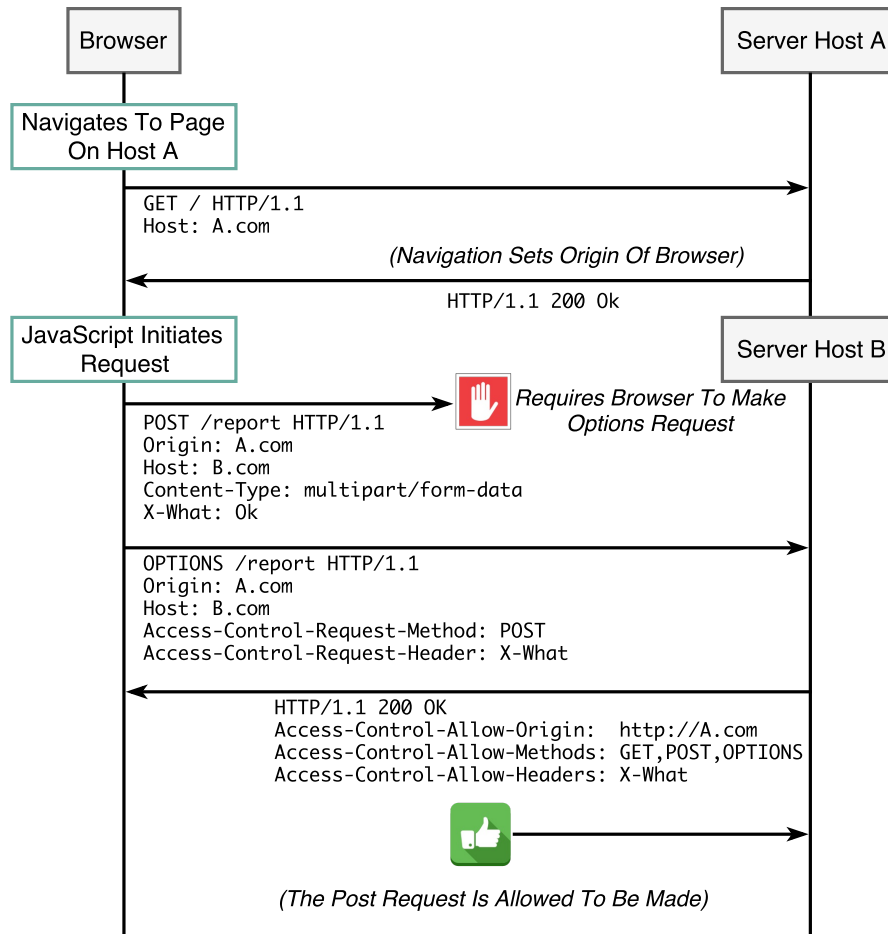


Fig. 18. Anatomy of a CORS request

Now the page at `awesome.example.com` may circumvent this process for making requests to `super.example.com` through using the JavaScript DOM API to set the `document.domain` to `example.com`. This is permissible because both `awesome` and `super` are suffixes of the origin `example.com` and changing the domain (origin) of the document to the shorter common prefix would no longer render requests to `super.example.com` from `awesome.example.com` governable by CORS. This also affects how the page at `example.com` could interact with any iframes it may embed from `super.example.com`, e.g. access the content that may be embedded inside of the iframe.

2.5 MEMENTO FRAMEWORK

The Memento Framework [34, 35, 36] defines inter-archive coordination, as well as provides succinct terminology for referring to archived resources and the inner workings of web archiving. In the thesis when we refer to web archiving we refer to it as the process by which a web resource becomes preserved at a given point in time. In the previous sections we discussed how 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” and a **URI-M** “is used to denote the URI of a **Memento**”, a memento represents “an Original Resource as it existed at time T” [35].

```
$ curl -I http://web.archive.org/web/http://grindtodeath.com/ -H
↪ 'Accept-Datetime: Wed, 31 Oct 2012 09:52:40 GMT'
> HEAD /web/grindtodeath.com/ HTTP/1.1
> Host: web.archive.org
> User-Agent: curl/7.58.0
> Accept: */*
> Accept-Datetime: Wed, 31 Oct 2012 09:52:40 GMT

> HTTP/1.1 302 FOUND
> Server: Tengine/2.2.2
> Date: Wed, 07 Mar 2018 22:15:53 GMT
> Content-Type: text/plain; charset=utf-8
> Content-Length: 32
> Connection: keep-alive
> Location: http://web.archive.org/web/20121031095240/http://grindtodeath.com/
> Vary: accept-datetime
> Link: <http://grindtodeath.com/>; rel="original",
↪ <http://web.archive.org/web/20121031095240/http://grindtodeath.com/>;
↪ rel="memento"; datetime="Wed, 31 Oct 2012 09:52:40 GMT",
↪ <http://web.archive.org/web/timemap/link/http://grindtodeath.com/>;
↪ rel="timemap"; type="application/link-format"
```

Fig. 19. Temporal content negotiation with the Internet Archive’s TimeGate to select the best Memento for the web page `http://grindtodeath.com/` at the datetime Wed, 31 Oct 2012 09:52:40 GMT

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. As shown in Figure 19, the Internet Archive’s TimeGate indicated it had found the best URI-M for the URI-R `http://grindtodeath.com/` at datetime Wed, 31 Oct 2012 09:52:40 GMT by sending an HTTP 302 Found response whose location HTTP header is set to the URI-M of selected memento. Also included in the response from

the TimeGate was the URI for the page's TimeMap (**URI-T**) found in the responses Link header. TimeMaps (Figure 20) for a URI-R contain a listing of the URI-Ms 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 relationship between mementos, TimeMaps, and the TimeGate are important as they provide a basis for this thesis, which is replaying (viewing) mementos of an archived web page using a browser.

```
$ curl http://web.archive.org/web/timemap/link/http://grindtodeath.com/
> GET /web/timemap/link/http://grindtodeath.com/ HTTP/1.1
> Host: web.archive.org
> User-Agent: curl/7.58.0
> Accept: */*

> HTTP/1.1 200 OK
> Server: Tengine/2.2.2
> Date: Wed, 07 Mar 2018 22:15:53 GMT
> Content-Type: application/link-format
> Transfer-Encoding: chunked
> Connection: keep-alive

> <http://www.grindtodeath.com:80/>; rel="original",
> <http://web.archive.org/web/timemap/link/http://grindtodeath.com/>; rel="self";
  ↪ type="application/link-format"; from="Fri, 20 May 2011 21:12:15 GMT",
> <http://web.archive.org>; rel="timegate",
> <http://web.archive.org/web/20110520211215/http://www.grindtodeath.com:80/>;
  ↪ rel="first memento"; datetime="Fri, 20 May 2011 21:12:15 GMT",
> <http://web.archive.org/web/20110621074527/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Tue, 21 Jun 2011 07:45:27 GMT",
> <http://web.archive.org/web/20110724010002/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Sun, 24 Jul 2011 01:00:02 GMT",
> <http://web.archive.org/web/20110824011507/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Wed, 24 Aug 2011 01:15:07 GMT",
> <http://web.archive.org/web/20110825033443/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Thu, 25 Aug 2011 03:34:43 GMT",
> <http://web.archive.org/web/20110902012722/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Fri, 02 Sep 2011 01:27:22 GMT",
> <http://web.archive.org/web/20110919001004/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Mon, 19 Sep 2011 00:10:04 GMT",
> <http://web.archive.org/web/20111001180937/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Sat, 01 Oct 2011 18:09:37 GMT",
> <http://web.archive.org/web/20111004172756/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Tue, 04 Oct 2011 17:27:56 GMT",
> <http://web.archive.org/web/20111104061223/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Fri, 04 Nov 2011 06:12:23 GMT",
> <http://web.archive.org/web/20111120224218/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Sun, 20 Nov 2011 22:42:18 GMT",
> <http://web.archive.org/web/20111201154328/http://www.grindtodeath.com/>;
  ↪ rel="memento"; datetime="Thu, 01 Dec 2011 15:43:28 GMT",
> <http://web.archive.org/web/20111205031921/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Mon, 05 Dec 2011 03:19:21 GMT",
> <http://web.archive.org/web/20120104221730/http://www.grindtodeath.com:80/>;
  ↪ rel="memento"; datetime="Wed, 04 Jan 2012 22:17:30 GMT",
> <http://web.archive.org/web/20120127115831/http://www.grindtodeath.com/>;
  ↪ rel="memento"; datetime="Fri, 27 Jan 2012 11:58:31 GMT",
> ...
```

Fig. 20. TimeMap for the web page <http://grindtodeath.com/> retrieved from the Internet Archive

When viewing an archived web page via the browser, the term replay encompasses both the browser auto-dereferencing the page plus embedded resources with the process the archive uses to resolve the URI-Ms contained within the page that the browser requests from the archive. Not only is the browser dereferencing the URI-M we wish to view, but also all other URIs contained in the representation of the URI-M. Encompassing this process is what Ainsworth et al. describe as a Composite Memento, “a root URI-M and all embedded URI-Ms required to recompose the presentation at the clients” [37]. In other words, composite mementos encompass the state and composition of a web page as it potentially existed at preservation time when accessed by the browser.

2.6 ARCHIVING AND REPLAY OF WEB PAGES

Web archives facilitate the replay of mementos. The oldest and most famous of these is the Internet Archive [8, 38]. Founded in 1996 by Brewster Kahle, the Internet Archive provided fundamental infrastructure for web archiving as a whole by providing the de facto means for the creation of mementos and the means to replay those mementos.

The Internet Archive uses Heritrix for its archival crawler [39]. Heritrix operates similarly to the browser in that it extracts all the URI-Rs contained within a web page [40] but without rendering of the page or execution of a page’s JavaScript. Heritrix generates a Web ARChive (WARC) file [41] which stores the HTTP requests and responses during crawling (Figure 22). WARC files generated by Heritrix or any other means are considered a snapshot of the web page as it existed at preservation time and, as discussed in the previous section (Section 2.5), constitutes a composite memento. The Internet Archives in turn makes these WARC files available for replay through the Wayback Machine.

The Internet Archive’s Wayback Machine [42, 43] is a combination front-end user interface to the contents of the archive and replay engine for viewing its archived web pages at a given point in time. The main page of the Wayback Machine (Figure 21a) allows users to enter a URI-R of web page in order to see if it is contained in the archive or select one of the promoted web pages to view. When a user enters in a URI to view, they are taken to a calendar view (Figure 21b) that displays the date of each memento, on which a user may click to view (Figure 21c).

Wayback Machine

INTERNET ARCHIVE

WayBackMachine

Enter a URL or words related to a site's home page

Explore more than 306 billion web pages saved over time

Tools

Wayback Machine Availability API
Build your own tools.

WordPress Broken Link Checker
Banish broken links from your blog.

404 Handler for Webmasters
Help users get where they were going.

Subscription Service

Archive-It enables you to capture, manage and search collections of digital content without any technical expertise or hosting facilities. Visit [Archive-It](#) to build and browse the collections.

Save Page Now

https://

SAVE PAGE

Capture a web page as it appears now for use as a trusted citation in the future.
Only available for sites that allow crawlers.

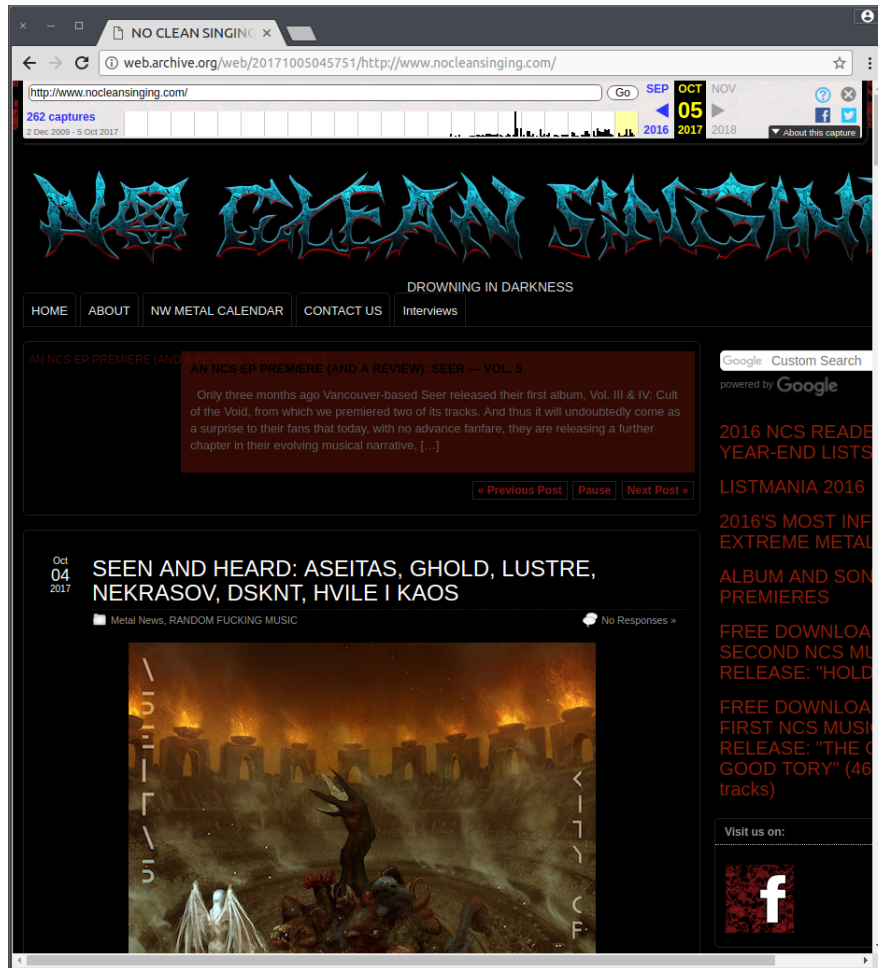
[FAQ](#) | [Contact Us](#) | [Beta Site Feedback](#) | [Terms of Service \(Dec 31, 2014\)](#)

The Wayback Machine is an initiative of the Internet Archive, a 501(c)(3) non-profit, building a digital library of Internet sites and other cultural artifacts in digital form. Other projects include Open Library & archive-it.org.

(a) Internet Archive Wayback Machine's main page

The screenshot shows the Wayback Machine interface for the website <http://www.nocleansing.com/>. The page features the Internet Archive logo and a search bar containing the URL. Below the search bar, it states "Explore more than 306 billion web pages saved over time" and "Saved 262 times between December 2, 2009 and October 5, 2017." A link to "Summary of nocleansing.com" is provided. A bar chart shows the frequency of saves from 2003 to 2017, with a significant peak in 2017. Below the chart is a calendar for the year 2017, with dates from January to September visible. Several dates are highlighted with colored circles: January 3, 9, 11, 27; February 3, 9, 11; April 3, 7, 16; June 1, 3, 6, 24, 27.

(b) Internet Archive Wayback Machine’s datetime selection



(c) Internet Archive Wayback Machine's Replay View

Fig. 21. Wayback Machine's interface For <http://www.nocleansinging.com/>

```

WARC/1.0
WARC-Type: request
WARC-Target-URI: http://example.com/
WARC-Date: 2017-10-03T21:32:53Z
WARC-Concurrent-To: <urn:uuid:69b8c0e0-a882-11e7-a534-7723e03cfee4>
WARC-Record-ID: <urn:uuid:69bc1c40-a882-11e7-a534-7723e03cfee4>
Content-Type: application/http; msgtype=request
Content-Length: 369

GET / HTTP/1.1
Host: example.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.38 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

WARC/1.0
WARC-Type: response
WARC-Target-URI: http://example.com/
WARC-Date: 2017-10-03T21:32:53Z
WARC-Record-ID: <urn:uuid:69bc9170-a882-11e7-a534-7723e03cfee4>
Content-Type: application/http; msgtype=response
Content-Length: 1577

HTTP/1.1 200 OK
Cache-Control: max-age=604800
Content-Type: text/html
Date: Tue, 03 Oct 2017 21:32:42 GMT
Etag: "359670651+gzip"
Expires: Tue, 10 Oct 2017 21:32:42 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (atl/FC90)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1270

<!doctype html>
<html>
<head>

```

Fig. 22. WARC file contents containing the HTTP requests and responses for the preserved web page <http://example.com>

2.7 URL REWRITING

This section discusses the current strategies employed by archival replay systems to ensure cohesive replay of an archived page and all of its embedded resources. This section will describe the methodology used by the open source implementation of the Wayback Machine, namely OpenWayback² and Pywb³, which are derived from the original closed source implementation used by the Internet Archive. Although there exist additional archival replay systems beyond those previously mentioned (e.g., PageFreezer⁴ and WebCitation) this section does not go into detail about their archival replay systems simply because they are closed source. URL rewriting is the

²<https://github.com/iipc/openwayback>

³<https://github.com/ikreymer/pywb>

⁴<https://www.pagefreezer.com/>

overall process for ensuring that the associated URIs for a memento (whether they be URIs to embedded resources or part of the archived HTTP headers) refer back to the archive on replay. For example, if we wish to view a memento for the URI-R `http://example.com` on “Sat, 07 Oct 2017 03:58:07 GMT” replayable from an archive at the URI `http://archive.com` we would use:

```
http://archive.com/20171007035807/http://example.com
```

Note that the actual URI-M used to replay a memento is dependent on the archives replay system. Now when we replay this memento, the natural assumption is that we will be viewing `http://example.com` at the Memento-Datetime and that any additional URI-Rs contained within the representation of the URI-M for `example.com` are also rewritten to include the time of the archived page. In order to rewrite URIs, archives rely on configurations that list what in the archived resources might contain URI-Rs that need to be rewritten. An example of this can be found in Figure 26, which lists the configuration used by OpenWayback’s standard attribute rewriter.

The standard attribute rewriter of OpenWayback is used to rewrite the URI-Rs contained in HTML element attributes to URI-Ms. As we can see in Figure 26, the standard attribute rewriter is targeting HTML elements that contain `src` and `href` attributes as well as element attributes that OpenWayback considers experimental. Take for instance the configuration file’s entry for the rewriting the link tags `href` for stylesheets:

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

You will notice that it does not explicitly define how the URI-R for

```
<link rel="stylesheet" href="...">
```

should be rewritten but rather it defines a modifier to be added, alongside the archival time of the replayed memento, acting as a hint for the archive indicating the specific rule based rewriter that should handle the rewriting of the resource (Figure 23).

```
<!-- pre rewritten -->
<link rel="stylesheet" href="/css/style.css">
<!-- post rewritten -->
<link rel="stylesheet" href="/20171007035807cs_/css/style.css">
```

Fig. 23. Pre And Post Rewritten Link Tags

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 corresponding stylesheets rewriter as used by Pywb is shown in Figure 24.

```

1  class CSSRewriter(RegexRewriter):
2      CSS_URL_REGEX =
3      • "url\\s*\\(\\s*(?:[\\\\"'"]|(?&.{1,4};))*\\s*( [^'\\"]+ )\\s*(?:[\\\\"'"]|(?
4      • &.{1,4};))*\\s*\\)"
5
6      CSS_IMPORT_NO_URL_REGEX = ("@import\\s+(?!url)\\(?(\\s*['\"]?\" +
7      • \"(?:url[\\s\\(\\)])([\\w.:/\\\\\\\\-]+)\"")
8
9      def __init__(self, rewriter):
10         rules = self._create_rules(rewriter)
11         super(CSSRewriter, self).__init__(rewriter, rules)
12
13     def _create_rules(self, rewriter):
14         return [(CSSRewriter.CSS_URL_REGEX,
15         • RegexRewriter.archival_rewrite(rewriter), 1),
16         (CSSRewriter.CSS_IMPORT_NO_URL_REGEX,
17         • RegexRewriter.archival_rewrite(rewriter), 1),
18         ]

```

Fig. 24. Pywb CSS regex rewriter

This rewriter defines two regular expressions (rules) that cover the combinations in which URIs can exist in CSS files (Figure 25). The regular expressions used by each respective rewriter must be applied to each and every 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 matches the current line, the rewriter will move on to the next until one of its rules matches or it reaches the end of file. This is the totality of the URL rewriting process and is the typical manner Wayback Machine implementations rewrite the URL-Rs to URI-Ms. It must be noted that non-Wayback Machine archives can do the rewriting differently or not at all (e.g., cached pages in search engines and historical MediaWiki pages). The URL rewriting process is an extremely important part of replay. If it was not for URL rewriting, the embedded resources of an archived web page would still point to the live web resulting in live web leakage or “zombies in the archive” [44].

```

1  @import "more.css";
2  @import url("evenMore.css");
3  body::before { content: url("someImage.png"); }
4
5  .backgroundIm { background-image: url("background.jpg"); }
6  .item { cursor: url("cur.svg"), auto; }
7  .item2 { border-image: url("pattern.svg") 40 40 repeat; }
8  div { background: url("it.png") }
9  ul { list-style: square url("redball.png"); }
10 ol { list-style-image: url("liImage.png"); }

```

Fig. 25. URIs in CSS files

```

1  # Default configuration for StandardAttributeRewriter
2  A.HREF.type=an
3  APPLET.CODEBASE.type=oe
4  APPLET.ARCHIVE.type=oe
5  AREA.HREF.type=an
6  BASE.HREF.type=an
7  EMBED.SRC.type=oe
8  IFRAME.SRC.type=if
9  IMG.SRC.type=im
10 IMG.SRCSET.type=ss
11 INPUT.SRC.type=im
12 FORM.ACTION.type=an
13 FRAME.SRC.type=fw
14 LINK[REL\=STYLESHEET].HREF.type=cs
15 LINK[REL\=SHORTCUT\ ICON].HREF.type=im
16 META[HTTP-EQUIV\=REFRESH].CONTENT.type=mt
17 META.URL.type=an
18 OBJECT.CODEBASE.type=oe
19 OBJECT.CDATA.type=oe
20 SCRIPT.SRC.type=js
21 SOURCE.SRCSET.type=ss
22 # Experimental
23 DIV.DATA-SRC.type=oe
24 DIV.DATA-URI.type=oe
25 LI.DATA-SRC.type=oe
26 LI.DATA-URI.type=oe
27 *.BACKGROUND.type=im
28 *.STYLE.type=ci
29 *.ONCLICK.type=jb
30 *.ONLOAD.type=jb
31 *.ONCHANGE.type=jb

```

Fig. 26. OpenWayback standard attribute rewrite configuration

2.8 SUMMARY

In this chapter, we discussed the concepts required to understand the remainder of this thesis. We looked at the roles that HTTP, HTML, and JavaScript have in the viewing of web pages using a web browser. We discussed the Memento protocol and how it ties into the replay of archived web pages. We then discussed the overall process of replaying archived web pages and how the Internet Archive's Wayback Machine functions at a high level. Finally, we discussed the URL rewriting process, which is critical for understanding chapters 4 and 5 of this thesis.

In the next chapter, we will discuss the work others have done that addresses the dynamic nature of the web.

CHAPTER 3

RELATED WORK

This chapter discusses previous work that has addressed the dynamic nature of the web and/or identified the effects of un-archived resources during replay.

3.1 ARCHIVING DYNAMIC CONTENT

In 2014 Mat Kelly created a test suite, the Archival Acid Test [45], for evaluating the ability of archival crawlers and replay systems for handling the more dynamic content of the web. The Archival Acid Test 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 of the test determined a crawler's ability to identify and handle URI variations such as relative (someFile.ext) or scheme-less URIs (//server.com) [21, 46]. The second category of tests examined how well the archival crawler could extract URIs from the JavaScript files which 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.

Tool \ Test	1a	1b	1c	1d	1e	1f	2a	2b	2c	2d	2e	2f	2g	2h	3a	3b	3c	3d
Archive.org	✗	.	.	.	✗	.
Archive.is	✗	.	.	.	✗	✗
Mummify.it	✗	.	.	✗	✗	.	.	✗	✗
Perma.cc	✗	.	.	✗	✗	.	✗	✗	✗
WebCite	✗	✗	✗	✗	✗	✗	✗	✗	✗	.	✗	✗	✗
Heritrix	✗	.	.	.	✗	✗
WARCreate	✗	✗	✗	✗	.	✗	✗
Wget	.	.	✗	✗	.	.	.	✗	✗	✗	.	.	✗

. = Test Passed ✗ = Test Failed

Fig. 27. Archival Acid Test tool evaluation results as of December 2014, [45, p. 3 Table 2]

Most archival crawlers and replay systems tested by Kelly (Figure 27) properly handled the first category of tests but only a few comprehensively handled the majority

of the tests. It was noted by Kelly that the tests all tools failed to pass were those involving a delay in resource retrieval. Brunelle et al. [47] conducted a study of 1861 URIs which had mementos from the Internet Archive between 2005 to 2012 in order to identify the impact JavaScript has on the archivability of web pages. In the study, Brunelle et al. 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 also observed that JavaScript was used by the mementos considered in the seven-year time span to load 33.7% of all resources, defining the resources which necessarily involved a delay in retrieval (by JavaScript or other means) as deferred representations. Similarly, Kelly et al. [48] showed the impact JavaScript has had on archivability of web pages over time, which was a loss of archived resources as JavaScript usage increased over time.

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. [49] 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 either missed by the archive's initial URL rewriting or dynamically generated by JavaScript back to the archive rather than the live web. The direct approach utilizes the client-side archival rewriting JavaScript library Wombat¹. Wombat is utilized by Pywb and Webrecorder² to override the JavaScript APIs of the browser to rewrite any unwritten URI-Rs into URI-Ms. Wombat includes a full URL rewriting system which utilizes overrides of the JavaScript Web and DOM APIs in order to rewrite any URIs which 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.

¹<https://github.com/ikreymer/pywb/blob/master/pywb/static/wombat.js>

²<https://webrecorder.io/>

3.3 SECURITY AND THE ARCHIVE

Replaying archived web pages that contain unarchived resources has brought up security concerns about how these unarchived resources could be used to alter the validity of the archived page. The leakage of live web resources into the replay of mementos is referred to as “zombies in the archive” [44, 50]. The term “zombies” is used to refer to live web resources that leak into replay due to the fact 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. Due to the lack of rewriting URI-Rs dynamically created by JavaScript on the part of the archive, when replaying a memento containing zombies, the zombie resources may give the false appearance that the archive has tampered or altered the memento and or its embedded resources.

Web archives do not or cannot preserve every resource for every page they archive [47, 48, 51, 52]. Due to this fact, archived web pages that are missing a portion of their embedded resources, when replayed, appear damaged. Brunelle, Kelly et al. [53, 54] looked at the proportion of missing resources for mementos in order to assess their damage, finding that the users’ perception of damage to be a more accurate metric for judging archival quality than the proportion of missing resources. This is an important finding considering that web archives do not preserve every resource of a preserved page and, depending on how damaged a memento is when replayed, may cause one to believe that the archive has tampered with the memento in some way.

Ainsworth et al. [37] looked at the temporal coherence of composite mementos, would make it appear as if the archive had maliciously modified the memento and its embedded resources, and found that they can exist in five states. 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 a 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.

Lerner et al. [55], unlike the others discussed in this section, describes three attacks

targeting web archives that are 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 archives). 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 his own.

The solutions posed by Lerner et al., namely archival modification of JavaScript at replay time and the separation of replayed content from the archives presentational components of replay, parallel the existing replay strategies employed by Webrecorder and Perma.cc [56].

3.4 SUMMARY

In this chapter, we discussed the previous work done by others' that addressed the dynamic nature of the web. We looked at how others have created tests for determining an archive's ability to replay JavaScript and looked at the studies conducted for measuring the impact JavaScript has had on web archiving over time. We then discussed two strategies for handling the reply of dynamic content and how the two strategies differ. Finally, we discussed security concerns surrounding the replay of un-rewritten content or lack their off and how this can be maliciously exploited. As discussed in this chapter, the need for securely handling the replay of dynamic content is becoming a necessity. In the next chapter, we will begin the discussion of how secure replay is accomplished.

CHAPTER 4

STYLES OF REPLAYING ARCHIVED WEB PAGES

In this chapter we will define terminology for describing the modifications a web archive applies to mementos in order to facilitate replay. We will also classify further modifications made to mementos based on the replay strategies employed by an archive.

4.1 ARCHIVAL LINKAGE MODIFICATIONS

In order 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” they were archived from but back to the archive (Section 2.7). We define the modifications made by the archive to a page and its embedded resources in order to serve (replay) them from the archive as **Archival Linkage Modifications**. Archival linkage modifications are the rewriting of URI-Rs contained within HTML, JavaScript, and CSS of an archived web page. Archival linkage modifications are not applicable to embedded resources such as PDFs [57] or the image, audio, and video media types [58] as they do not contain URLs which can be used to fetch additional resources.

4.1.1 LINKAGE MODIFICATIONS: 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 [12].

The HTML specification explicitly defines six attributes, with corresponding elements that may be used to embedded resources into the page or link to another page or resource (Table 3).

Table 3
HTML elements containing the `src` and `href` attributes

Attribute	HTML Elements
href	a, area, base, link
	audio, embed, script, img
src	input, frame, iframe, source
	track, embed video
poster	video, audio
srcset	img, source
action	form
data	object

Of the six attributes shown in Table 3, the *href* and *src* are associated with the more commonly used HTML elements. The *src* attribute of the eleven elements seen in Table 3 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, unlike the *src* attribute which has only one purpose, has three purposes based on the semantics of the associated tag. The first purpose is to provide navigable links within the document’s text through an `a` tag or a region on an image that is also a navigable link using the `area` 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 values of the *src* or *href* attribute which are relative URIs are resolved by using the `base` tag. 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` tags *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` tags indicates how to interpret the link (*href*) this tag is making to another resource. For example, if there exist different URIs to a single page, a common occurrence for e-commerce web sites, the usage of *rel=canonical* will indicate to search engines indexing the site that the value provided by the *href* is the definitive link to the current page. Using *rel=canonical* [59] does not require the browser to fetch an additional resource for the page because the value for *rel* and *href* of the `link` tag only defines a relationship. The rel types listed in Table 4 indicate the only necessary modification (rewriting of) the `link` tag's *href* because they necessarily initiate a browser fetch whereas the other rel types do not [60]. Unlike the explicitly defined relationship for the attributes displayed in Table 3, the HTML specification defines custom attributes that may be added to any arbitrary tag.

Table 4
Link Rel Types Requiring A Browser Resource Fetch

Rel Type	Indicates
stylesheet	Indicates the resource linked to is a stylesheet and requires the browser to fetch it
prefetch	Requests the browser fetch the resource in advance as it may be potentially used by the current page or on another page on the current domain
preload	Requires the browser to fetch the resource immediately as it will be used by the current page
icon	Indicates a resource which represents the page to be used in a browser tab
dns-prefetch	Requires the browser to perform a DNS lookup and protocol handshaking for the resource
manifest	The URI-R of the link tag is a web app manifest
import	Import a document fragment into the current document

Custom attributes, as defined by the HTML specification, are attributes that

start with “data-” or are any combination of non-space ASCII characters and may contain any value permissible for HTML attributes [12, 25]. A contrived example demonstrating the use of custom attributes can be seen in Figure 28, showing how they are used by a page’s JavaScript (lines 9-10), by its CSS (lines 13), and its use on custom elements [25] which are purely JavaScript powered (lines 3-20). The definition of the custom element is associated (registered) with the tag (line 20) by registering the name of the custom tag to the constructor of the JavaScript definition for the custom element [12]. Archives must carefully consider each and every custom attribute of tags even for those that are not explicitly defined in the HTML specification in order to preserve any potential embeds a custom element may add.

```

1 <cool-image-bro superSizeMe="forSure"
  • image-me-bro="http://lorempixel.com/400/200/"></cool-image-bro>
2 <script>
3   class CoolImageBro extends HTMLElement {
4     constructor () {
5       super()
6       const shadow = this.attachShadow({mode: 'open'})
7       const im = document.createElement('img')
8       im.style = 'display: block; float: left;'
9       im.src = this.getAttribute('image-me-bro')
10      const upsize = this.getAttribute('superSizeMe')
11      if (upsized && upsize === 'forSure') {
12        const style = document.createElement('style')
13        style.innerHTML = "img[largeCoke='sideOfFries'] { width: 100%; height: 100%; }"
14        shadow.appendChild(style)
15        im.setAttribute('largeCoke', 'sideOfFries')
16      }
17      shadow.appendChild(im)
18    }
19  }
20  window.customElements.define('cool-image-bro', CoolImageBro)
21 </script>

```

Fig. 28. Custom HTML element using custom attributes

4.1.2 LINKAGE MODIFICATIONS: JAVASCRIPT

JavaScript is a dynamic, interpreted programming language that requires execution in order to determine the ultimate result of the values the code operates on or produces. This requires archives seeking to apply linkage modifications to the JavaScript code for an archived page to carefully consider the context in which an URI-R may appear in the JavaScript code along with the *how* and *where* they may be retrieved from by the archived JavaScript. Figure 29 shows two examples where it is safe for an

archive to rewrite URI-Rs (lines 2,4) and three examples of when it is impossible to do so (lines 7, 9-12, 14-20). The examples that are rewritable do not involve any dynamically computed parts, whereas those that do involve dynamically computed values which are not easily discoverable and thus are not rewritable.

```

1 // identifiable URI-Rs
2 const schemeLessURL = '//www.google-analytics.com/analytics.js'
3
4 $('#footerContainer').append($('
&amp;amp;amp;lt;p&amp;amp;amp;gt;Fig. 29. Live web JavaScript usage of URI-Rs&amp;amp;amp;lt;/p&amp;amp;amp;gt;
&amp;amp;amp;lt;/div&amp;amp;amp;gt;
&amp;amp;amp;lt;div data-bbox="171 574 883 686" data-label="Text"&amp;amp;amp;gt;
&amp;amp;amp;lt;p&amp;amp;amp;gt;The un-rewritable nature of dynamically computed URI-Rs, especially those shown in Figure 29, shows that JavaScript is syntax and interpretation dependent (line 11). Humans can tell by inspection that “n.src” is associated with the JavaScript DOM API for the &amp;amp;amp;lt;code&amp;amp;amp;gt;script&amp;amp;amp;lt;/code&amp;amp;amp;gt; tag created within the function, but it may as well have been the &amp;amp;amp;lt;code&amp;amp;amp;gt;src&amp;amp;amp;lt;/code&amp;amp;amp;gt; attribute of some object not associated with a DOM element.&amp;amp;amp;lt;/p&amp;amp;amp;gt;
&amp;amp;amp;lt;/div&amp;amp;amp;gt;
&amp;amp;amp;lt;div data-bbox="171 692 883 876" data-label="Text"&amp;amp;amp;gt;
&amp;amp;amp;lt;p&amp;amp;amp;gt;Now consider Figure 30, which shows how a script is used to embed JSON into a page, and Figure 31, showing the JavaScript code used to make the actual request. The URI-R inside the embedded JSON is clearly identifiable and can be rewritten, whereas the URI-R created by the JavaScript code performing the actual request (Figure 31, lines 3-6) cannot. In this case, the URI-R contained in the embedded JSON (Figure 30) was rewritten and the request was made to the archive instead of the live web. The archive would now have to consider that the response for the request, as seen in Figure 32, contains more JSON with URI-Rs that are rewritable.&amp;amp;amp;lt;/p&amp;amp;amp;gt;
&amp;amp;amp;lt;/div&amp;amp;amp;gt;
```



```

<script id="flik" type="application/json">
{"apiKey":"3e75fa0fe8a8ea56a70bfb66a53e9220","meth":"GET","url":"https://api.flic
↪ kr.com/services/rest/?method=flickr.people.getPublicPhotos&user_id=32951986%4
↪ 0N05&extras=url_q&format=json&nojsoncallback=1&api_key="}
</script>

```

Fig. 30. Script tag embedding JSON

```

1 function loadPhotosJQ () {
2   return new Promise((resolve, reject) => {
3     let pjs = $.parseJSON($('#flik').text())
4     let {apiKey, meth, url} = pjs
5     url = url + apiKey
6     $.ajax(url, { type: meth, dataType: 'json', jsonp: false })
7       .done((data, textStatus, jqXHR) => resolve(data.photos.photo))
8       .fail((jqXHR, textStatus, errorThrown) => reject(errorThrown))
9   })
10 }
11

```

Fig. 31. Request made to Flickr API using embedded JSON

```

{"photos":{"page":1,"pages":26,"perpage":100,"total":"2525","photo":[{"id":"4058784908","owner":"3295
1986@N05","secret":"401c422f0d","server":"2585","farm":3,"title":"The supreme triumph of a popular
song -- its hand-organ appe...", "ispublic":1,"isfriend":0,"isfamily":0,"url_q":"https://farm3.stati
cflickr.com/2585/4058784908_401c422f0d_q.jpg","height_q":"150","width_q":"150"}]},"stat":"ok"}

```

Fig. 32. Response body for the GET request to Flickr API

4.1.3 LINKAGE MODIFICATIONS: CSS

The final component of archival linkage modifications is modifications made to CSS. Archival linkage modifications made to CSS are trivial in comparison to the linkage modifications made to HTML or JavaScript as URI-Rs can exist in CSS only two ways (Figure 33): using the “@import” keyword followed either 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.

```

@import "nihility.css"
@import url("http://cssHeaven.com/angelic.css")
@font-face {
  font-family: 'TheDude';
  src: url('Abides.woff2') format('woff2');
}

```

Fig. 33. URIs in CSS files

4.2 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. Specifically, these modifications only negate HTML element and attribute pairs that cause the browser to navigate away from the URI-M (Figure 34), apply a security policy not originating from the archive (Figure 35), or to embedded a hash of the original representation for resources loaded by the `script` and `link` tags (Figure 40). The first of these modifications is the negation of `meta` tags that alters the URI-M of the currently replayed page and refreshes the browser, causing navigation to the altered URI-M [50].

```

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

```

Fig. 34. Meta refresh tag

`Meta` tags that refresh the browser use a HTTP header that never existed “refresh” and were removed from the fourth revision of the HTML specification [61] but then added again in the fifth revision [25] due to widespread usage and browser support. The meta refresh tag seen in Figure 34 defines in the value for the *content* attribute a wait time of thirty-five seconds before the browser will refresh the page appending “?refresher=666” to the current URI of the browser, creating a new URI. When a browser refresh occurs and the URI has changed, this causes navigation to the new URL, which is not desired when viewing the page in the archive. In the case of our example, the new URI navigated to by the browser is the same URI with an added query parameter, but could well have been a completely different URI or the same one. This is a commonly used trick by news sites that wish to update the contents

of their page without relying on JavaScript. But when such data is archived, this will change the URI-M of the page to one the archive may not have, simply because of the added query parameter and the fact that this live web “locally understood” canonicalization cannot be built into archive replay systems. 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. However, if this behavior is desired, then the URL specified in the value of the content attribute should not be rewritten.

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 [62] for the page it should be applied to. The Internet Archive, as a direct response to the paper by Lerner et al. [55] (discussed in Chapter 3), is now applying their own content security policy during replay. Content security policies 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 [63]. The issue arises when `meta` tags are used to define one or more policies for a page. Meta tag delivered policies are additive to any CSP delivered in the HTTP of the response for the page (Figure 35).

```
<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. 35. Content Security Policy defined in a meta tag

The `meta` tag in Figure 35 defines a policy using the directives in Table 5 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 the replay origin is the archives, not the one listed in the policy. For example, consider the archived page replayed from the Internet Archive seen in Figure 37, which was created to demonstrate the affects of a meta tag delivered content security policy. The policy it

delivers is similar to the one found in Figure 35 except that `http://mydomain.com` was replaced with the domain of the page (`http://cs.odu.edu`) and the full URL to the page (`http://cs.odu.edu/~jberlin/simpleMetaCSP.html`).

Because `meta` tag delivered content security policies are additive and applied after any policy that maybe delivered via HTTP headers, the browser refuses to load both the embedded resources for the page and the Internet Archive banner (Figures 37 and 39). But as seen in Figure 37 the page was preservable using the save page now feature of the Wayback Machine even though the `meta` tag delivered content security policy blocked the Wayback Machine's control of the page (Figure 38). To better understand how this was accomplished, consider the annotated HTML (Figure 36) of the page delivering a content security policy via a meta tag. Because the content security policy delivering `meta` tag (line 11) comes before the Wayback Machine injected banner assets (lines 17-23), the `meta` tag defined policy is applied to the injected banner assets plus the pages own embedded resources after the Internet Archives own policy, thus causing the browser to block them. However, embedded resources loaded by resources which precede the `meta` CSP tag (lines 2-9) after the `meta` CSP tag has been parsed by the browser will be blocked as well [62]. The *nonce* attribute used by the page (Figure 36 lines 14, 26-29) is a cryptographic number used once (nonce) and is used to white list `style` and `script` tags, that is to say ensure that they will not be blocked by the content security policy [12, 62]. 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. 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 (Figure 40).

```

1 <!-- nested tags and tag contents removed for presentation -->
2 <script type="text/javascript" src="/static/js/analytics.js?v=1519377156.0" charset="utf-8"></script>
3 <script type="text/javascript"></script>
4 <script type="text/javascript" src="/static/js/wbhack.js?v=1519377156.0" charset="utf-8"></script>
5 <script type="text/javascript"></script>
6 <link rel="stylesheet" type="text/css" href="/static/css/banner-styles.css?v=1519377156.0"/>
7 <link rel="stylesheet" type="text/css" href="/static/css/iconocheive.css?v=1519377156.0"/>
8 <!--End Wayback Rewrite JS Include-->
9 <meta charset="UTF-8">
10
11 <meta http-equiv="Content-Security-Policy" content="default-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4'
  * 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html;
  * connect-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7'
  * http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; frame-src 'nonce-hahahah1' 'nonce-hahahah2'
  * 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu
  * http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; img-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4'
  * 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html;
  * manifest-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7'
  * http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; media-src 'nonce-hahahah1' 'nonce-hahahah2'
  * 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu
  * http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; object-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4'
  * 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html;
  * script-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7'
  * http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; style-src 'nonce-hahahah1' 'nonce-hahahah2'
  * 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu
  * http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; font-src data: 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3'
  * 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu
  * http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html; worker-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4'
  * 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' http://www.cs.odu.edu http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html;">
12 <meta name="viewport" content="width=device-width, initial-scale=1.0">
13 <title>Simple Meta CSP</title>
14 <style nonce="hahahah1"></style>
15 </head>
16 <body><!--BEGIN WAYBACK TOOLBAR INSERT-->
17 <script type="text/javascript" src="/static/js/timestamp.js?v=1519377156.0" charset="utf-8"></script>
18 <script type="text/javascript" src="/static/js/graph-calc.js?v=1519377156.0" charset="utf-8"></script>
19 <script type="text/javascript" src="/static/js/auto-complete.js?v=1519377156.0" charset="utf-8"></script>
20 <script type="text/javascript" src="/static/js/toolbar.js?v=1519377156.0" charset="utf-8"></script>
21 <style type="text/css"></style>
22 <div id="wm-ipp" lang="en" style="display:none;direction:ltr;"></div>
23 <script type="text/javascript"></script>
24 <!--END WAYBACK TOOLBAR INSERT-->
25 <div id="container"></div>
26 <script nonce="hahahah2"></script>
27 <script nonce="hahahah3"></script>
28 <script nonce="hahahah4"></script>
29 <script nonce="hahahah5"></script>
30 </body>

```

Additional Resources Loaded From Within These Files
After Meta CSP Tag Parsed By Browser Bocked

All Resource Loaded
Are Completely Bocked

Fig. 36. The HTML of the page delivering a Content Security Policy via meta tag with the areas affected by the policy during replay from the Internet Archive's Wayback Machine highlighted. <http://web.archive.org/web/20171003192253/http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>

Simple Meta CSP

John

web.archive.org/web/20171003192253/http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html

http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html

Go

3 captures

29 Sep 2017 - 03 Oct 2017

Previous capture 03 2016

Next capture 2018

SEP OCT NOV

2016 2017 2018

COLLECTED BY

Organization: **Internet Archive**

The Internet Archive discovers and captures web pages through many different web crawls. At any given time several distinct crawls are running, some for months, and some every day or longer. View the web archive through the Wayback Machine.

Collection: **Live Web Proxy Crawls**

Content crawled via the Wayback Machine Live Proxy mostly by the Save Page Now feature on web.archive.org.

Liveweb proxy is a component of Internet Archive's wayback machine project. The liveweb proxy captures the content of a web page in real time, archives it into a ARC or WARC file and returns the ARC/WARC record back to the wayback machine to process. The recorded ARC/WARC file becomes part of the wayback machine in due course of time.

TIMESTAMPS

loading

A Document may deliver a Content Security Policy via one or more HTML meta elements whose http-equiv attributes are an ASCII case-insensitive match for the string "Content-Security-Policy"

Authors are strongly encouraged to place meta elements as early in the document as possible, because policies in meta elements are not applied to content which precedes them. In particular, note that resources fetched or prefetched using the Link HTTP response header field, and resources fetched or prefetched using link and script elements which precede a meta-delivered policy will not be blocked.

The nonce attribute provides a way to whitelist specific inline script and style elements, without using the CSP unsafe-inline directive

Nonces tell the browser that the contents of the inline element are not malicious but put in the document by the entity that served the document

Fig. 37. Meta Tag Delivered Content Security Policy preventing Internet Archive From Loading Wayback Machine Resources. <http://web.archive.org/web/20171003192253/http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>

Developer Tools - <http://web.archive.org/web/20171003192253/http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>

Elements Console Sources Network Performance Memory Application Security Audits

top Refused to load Default levels Group similar 39 hidden

- Refused to load the script '<http://web.archive.org/static/js/timestamp.js?v=1518460053.0>' because it violates the following Content Security Policy directive: "script-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the stylesheet '<http://web.archive.org/static/css/record.css>' because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the script '<http://web.archive.org/static/js/graph-calc.js?v=1518460053.0>' because it violates the following Content Security Policy directive: "script-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the script '<http://web.archive.org/static/js/auto-complete.js?v=1518460053.0>' because it violates the following Content Security Policy directive: "script-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the script '<http://web.archive.org/static/js/toolbar.js?v=1518460053.0>' because it violates the following Content Security Policy directive: "script-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the image '' because it violates the following Content Security Policy directive: "img-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <URL> <URL>".
- Refused to load the font '' because it violates the following Content Security Policy directive: "font-src data: 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <URL> <URL>".
- Refused to load the stylesheet '<http://web.archive.org/static/css/banner-styles.css?v=1518460053.0>' because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the stylesheet '<http://web.archive.org/static/css/record.css>' because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".
- Refused to load the stylesheet '<http://web.archive.org/static/css/iconoviche.css?v=1518460053.0>' because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>".

Developer Tools - <http://web.archive.org/web/20171003192253/http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>

Elements Console Sources Network Performance Memory Application Security Audits

top inline Default levels Group similar 8 hidden

- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-Xx7GyLwRs4JEfBLjI120zxIGUE7dmv+qYfpe47fX8='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-NN4ISvuokJ4v5ZVRenH0dK13ITHOuJMa176kdRB2Y='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-1YE4fENdLFEBRvJfEhRlBb0a163JevJNMCK12gV='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-1YwYh1McsGmCuZLpEz2N5dINr1qk1sLbLhEcqM='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-MigJ00kRMMv75FWKvVbUjukPaRvm386C090CIeF8I='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-BtFkuTJezZaKf8YqTxnX0ul9r9PfZj25FjngNGwPLKQ='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-0rQM20G0pFu78RQD0vXwbZevI1BDAYrdnpRh1D2H0='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-0HFZ5RmP1nqF310h1Vg3gM+2KSNFAoaSN0dFUp5+I='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-hFp3FrAiaQTxfjB00dV/YaZa42t/R2z1xhUab/hq9Y='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-jtmIdJ8IHbS18BaR6B0MOU1x7gAka5Lz1h0F5D00='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-m3XT1F20AAL/3olhZCLpVDDCC+0hIppq95Z300k='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-Vv7WtUbsu2D+kFu36NBc05J7a85M6Cv2Ahq7V7are4='), or a nonce ('nonce-...') is required to enable inline execution.
- Refused to apply inline style because it violates the following Content Security Policy directive: "style-src 'nonce-hahahah1' 'nonce-hahahah2' 'nonce-hahahah3' 'nonce-hahahah4' 'nonce-hahahah5' 'nonce-hahahah6' 'nonce-hahahah7' <http://www.cs.odu.edu> <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>". Either the 'unsafe-inline' keyword, a hash ('sha256-Vv7WtUbsu2D+kFu36NBc05J7a85M6Cv2Ahq7V7are4='), or a nonce ('nonce-...') is required to enable inline execution.

Fig. 38. Blocked Wayback and archived embedded resources

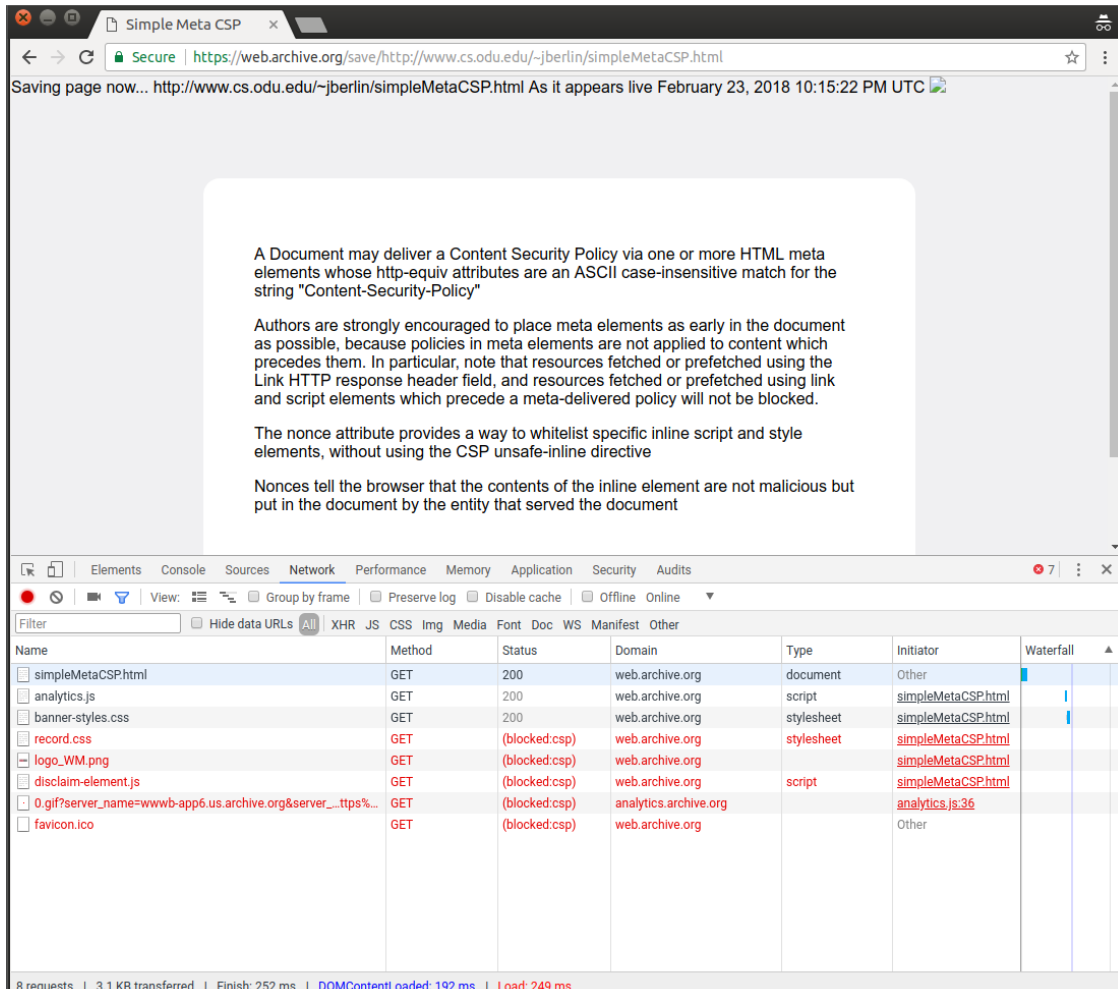


Fig. 39. Meta tag delivered content security policy preventing Internet Archive from controlling save page now. <http://www.cs.odu.edu/~jberlin/simpleMetaCSP.html>

The *integrity* attribute of the `link` and `script` tags shown in Figure 40 consists of a hash used to validate the hash computed by the browser upon receiving the response body of the resource [64]. 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 contents (Sections 2.7 and 4.1), 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.


```

<link rel="stylesheet" href="theStyleSheet.css" integrity="sha384-eKdwSs2g6PL_
→ +F9/RnQ14sov7h5SAFYgq8WJln2tXHOSW/7fJt4G+Td7PcVzkJunk">
<script src="theScript.js"
→ integrity="sha256-qerliYS7q6jrFLa4BJJ3g1ua00vkPz9SjuYsHPLpVzE="></script>

```

Fig. 40. Integrity attribute of the script and link tags

Table 5
Content Security Policy Directives

Directive	Restricts
default-src	default origin
script-src	JavaScript
style-src	stylesheets
img-src	images
font-src	fonts
object-src	plugins
media-src	audio and video
frame-src	frames or iframes
worker-src	service or web workers
connect-src	JavaScript HTTP Requests

4.3 SANDBOXED REPLAY

Sandboxed replay is the style of replay that separates the replayed page from the archive-controlled portion of the page through isolation. The term sandboxing in the case of replay is similar to the architectural design of the Chromium browser [65, 66]. The Chromium browser separates the browser kernel (e.g., network and filesystem stacks) from the rendering engine (e.g., HTML and CSS parsing, image decoding and JavaScript engine). Chromium’s rendering engine is run with restricted privileges (i.e., in a sandbox) alongside the browser kernel which treats its counterpart as a black box. For any page to gain access to the outside world (i.e., HTTP requests), it

must first be allowed to do so by the browser kernel. Separation of the browser kernel and rendering engine is a direct reflection of the base assumption the Chromium development team that the rendering engine is always compromised [66]. Archives and replay systems that employ this replay paradigm share this assumption, but rather than assuming the render is always compromised, it is that the pages being replayed are always compromised.

4.3.1 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 Webrecorder¹ (Figure 41). The two green boxes in Figure 41 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 simply highlights additional elements used in conjunction with the navbar. These elements are on the origin <https://webrecorder.io>, unlike the actual memento being replayed, which is embedded using an iframe from <https://wbrc.io>, highlighted by the red box. Iframes embed one web page in another using *a browser context* i.e., environments in which document objects are presented to the user [12]. Because iframes bring in what is essentially another browser window into the current one, they have their own security restrictions that is used by the archive to achieve replay isolation.

¹<https://webrecorder.io/jberlin/beautify-js-sites/20170630215424/http://2016.makemepulse.com/>

```

<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="javascrip...></script>
          <script>...</script>
          <script src="javascrip...></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. 41. Example of replay isolation's sandbox on a page replayed by webrecorder.io. The sandboxed portion is outlined in red.

When the embedded page is from a different origin [12] than the embedding page, the content brought in has limited access to the embedding page and vice versa. What the specification means by limited access to the embedded page is that interaction between the two pages occurs solely through message passing only. Direct access to the contents of the embedding page by the embedded page is disallowed by the security features of the browser. 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.

The separability inherent to replay isolation can be more easily seen when viewing the browser provided frame tree (Figure 42). Frame trees are a frame organized representation of a page's resources. The web page's main document is represented

as the `top` frame and each child frame is an embedded `iframe` contained in the main page (`top frame`) or some other frame. Consider the frame tree in Figure 42 for replay of the `http://2016.makemepulse.com` memento. The non-replay archive added portion of replaying web pages exists in the top frame on host `webrecorder.io` (green box), whereas the actual memento being replayed exists inside the top frame as the frame named `replay_iframe` (red box) on host `wbrc.io`. Even though the archived-controlled portion of replay is embedding the replayed memento on `webrecorder.io`, replay isolation occurs because the memento is replayed from the host `wbrc.io` not `webrecorder.io`. From the user's point of view (Figure 43), replay appears to happen on `webrecorder.io`, as the archive-controlled portions (green box) and the replayed page (red box) exist together, but in reality they are isolated.

Providing the same replay experience as if replay separation was not in place requires that modifications be made to both the embedded resources of the replayed page and the JavaScript environment that runs the archived JavaScript.

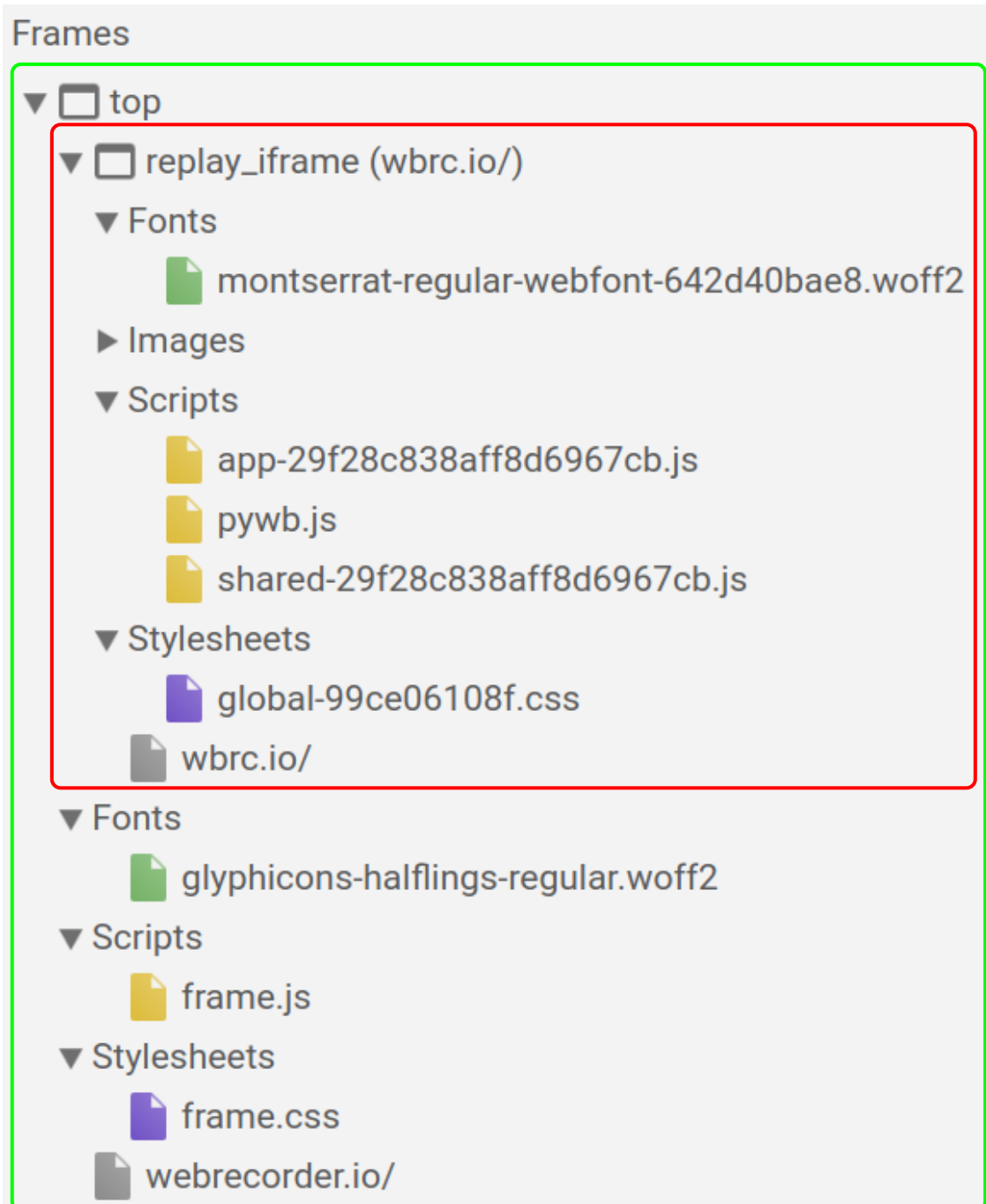


Fig. 42. Sandboxing Replay frame tree. The green box represents the webrecorder.io domain and the red box represents the wbrc.io domain

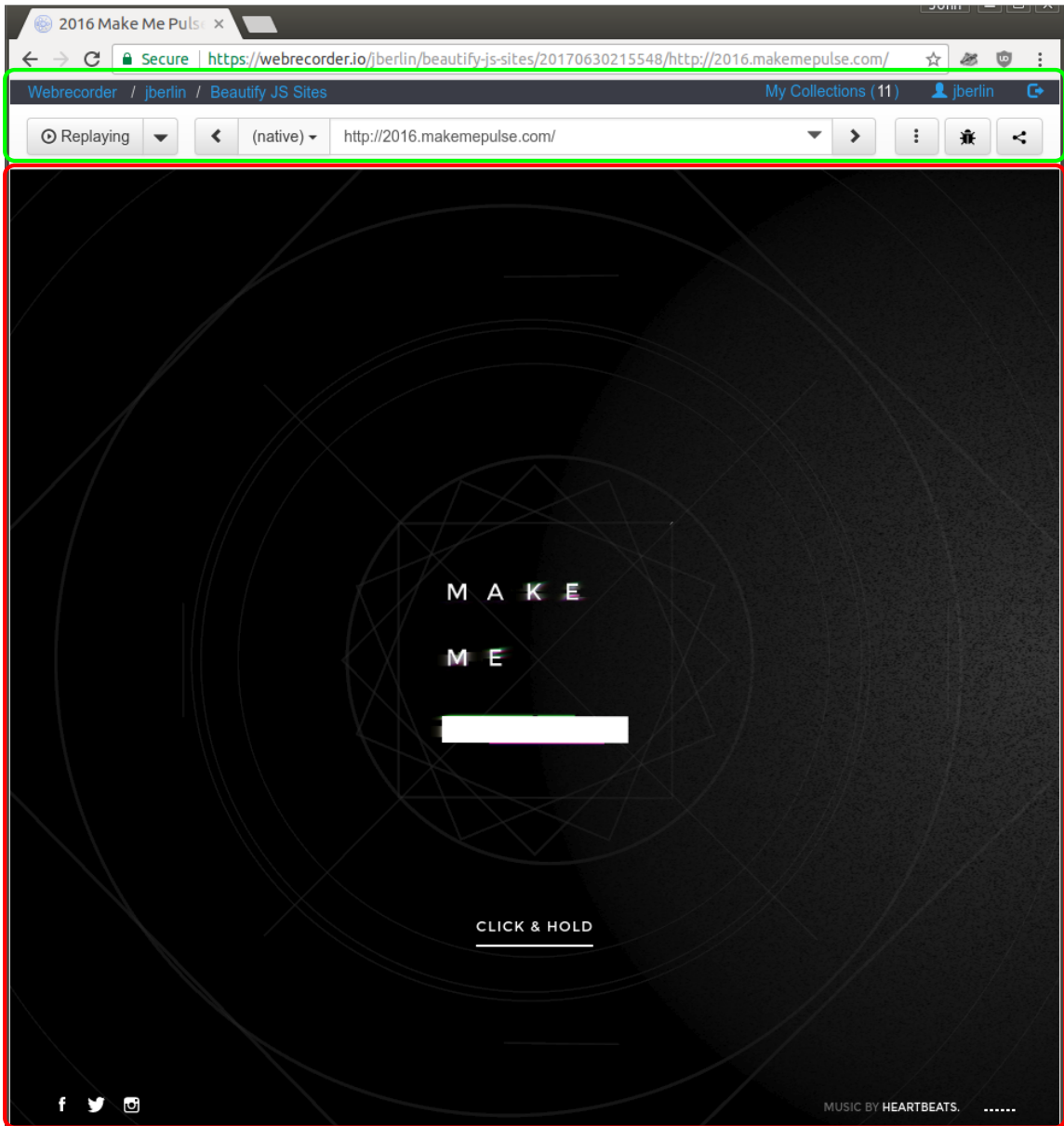


Fig. 43. Replay Isolation user view. The green box represents the necessarily archive controlled portion of replay and the red box represents the replayed page

4.3.2 TEMPORAL JAILING

Even though an archive may solely employ **replay isolation**, the archived JavaScript of the replayed page still has the ability to reach out to the live web [55, 56]. When that happens, the sandbox the archive was attempting to create through **replay isolation** and URL rewriting was escaped, thus causing loss of control over replay on the part of the archive. This is undesirable by the archive only using **replay isolation** but even more so on the part of the archive using **sandboxed replay**. Because of this, the archive using **sandboxed replay** goes a step beyond URL rewriting and **replay isolation** by its usage of **temporal jailing**.

Temporal jailing is the emulation of the JavaScript environment as it existed at the original memento-datetime through client-side rewriting and archive navigational control over the page. Temporal Jailing is in essence a total conversion of the JavaScript environment (APIs) made available by the respective browser replaying the page. Total conversion by necessity implies modification that can be broken down into three categories based on the definition of Temporal Jailing.

The first category is **emulation modifications**. Emulation modifications ensure the archived JavaScript, when replayed, cannot detect that the page is being replayed and the behavior/function of the archived page persists post-archiving. Figure 44 displays a few ways one could detect if a page is contained in an iframe.

```

window !== window.top           window.self !== window.top
window.parent.frames.length > 0  window.frameElement !== null
parent !== top                   self !== top

```

Fig. 44. Ways in which JavaScript may detect it is being replayed inside an iframe

JavaScript frameworks typically include these kinds of checks in order to generalize the use cases for which they may be used or by the creators of web pages to ensure that the page cannot be embedded in another website. Even a rudimentary search of the most popular open source code repository hosting service GitHub (Figure 45) shows use 91,063 results for `self !== top`, with the second and third results changing the location of the page if the code detects it is within an iframe.

The screenshot shows a GitHub search interface with the query "self !== top" in the search bar. The search results are sorted by "Best match" and show 91,063 available code results. Three results are displayed:

- AdeTheux/Zendesk-widgets – hide_header_for_KB_sidebar_app.js** (JavaScript):


```

1  if (self !== top)
2  {
3  $('#top').remove();
4  $('#page').css('maxWidth', '100%');
5  $('html, body').css('background-color', '#FFFFFF');
6  }

```
- milo-du/kuaixun-publisher-admin – page.js** (JavaScript):


```

1  (function(self) {
2      if( self == self.top ) {
3          self.location.href='/index.html';
4      }
5  })(window);

```
- ChangMM/tucao.hustonline.local – index.js** (JavaScript):


```

1  /**
2   * Created by cmm on 16/3/19.
3   */
4  if(top!==self)
5      top.location.href=this.location.href;

```

Fig. 45. Frame busting JavaScript Github search

Needless to say this is an unwanted behavior for archives using Sandboxed replay, which is why the JavaScript APIs listed in Figure 44 must be overridden to only indicate a page is in an iframe if it is a sub-page contained in an iframe within the replayed page.

Similar to this is the emulation of the JavaScript date object, which if not overridden through no fault of the archive, will necessarily expose the current date and time the replayed page is being viewed on. This happens only because the global JavaScript date object is tied to the system clock which the browser uses in its creation. If the date object was not overridden, then archiving a page that displays a specific message or provides a service at a specific date or time would

not be preserved simply due to the archive not ensuring the JavaScript date object correctly reflects the replay time. Another aspect of the date object is that it is commonly used in URI creation and requests made by the archived JavaScript. The resources the archived page requests are tied to the specific URI-Rs made at archival time. Unless the date object used in the creation of the URI-M correctly reflects the original memento-datetime, the responses for requests made with the incorrect datetime will receive a HTTP 404 response even though the archive does indeed contain the resource requested but had indexed it to the URI created at archival time.

The final primary **emulation modifications** are the location exposing overrides. Location exposing overrides required for **emulation modifications** also intersect and are necessarily included in the next class of modifications, which are **navigational modifications**. Since the archive uses **sandboxed replay** which necessarily separates the presentational components of replay from the replaying of the page itself (**replay isolation**), handling JavaScript initiated navigation requires overriding those APIs (Figure 46).

```

document.location  window.location
window.open        window.history
document.domain    location

```

Fig. 46. Location exposing and navigation control JavaScript APIs

When overridden by the archive, the JavaScript APIs listed in Figure 46 would 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 [11], but 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. This means that the URIs used to cause navigation were rewritten to URI-Ms (Figure 47 lines 570-574, 576-581, 585).

```

568 function WombatLocation (orig_loc) {
569   this._orig_loc = orig_loc;
570   this.replace = function (url) {
571     var new_url = rewrite_url(url);
572     var orig = extract_orig(new_url);
573     if (orig == this.href) { return orig; }
574     return this._orig_loc.replace(new_url);
575   }
576   this.assign = function (url) {
577     var new_url = rewrite_url(url);
578     var orig = extract_orig(new_url);
579     if (orig == this.href) { return orig; }
580     return this._orig_loc.assign(new_url);
581   }
582   this.reload = function () { return this._orig_loc.reload(); }
583   this.orig_getter = function (prop) { return this._orig_loc[prop]; }
584   this.orig_setter = function (prop, value) { this._orig_loc[prop] = value; }
585   init_loc_override(this, this.orig_setter, this.orig_getter);
586   set_loc(this, orig_loc.href);
587   this.toString = function () { return this.href; }
588   for (var prop in orig_loc) {
589     if (this.hasOwnProperty(prop)) { continue;}
590     if ((typeof orig_loc[prop]) != "function") {
591       this[prop] = orig_loc[prop];
592     }
593   }
594 }

```

Fig. 47. Replay Isolation location override implementation

The final class of modifications necessary for sandboxed replay is client-side rewriting modifications. **Client-side rewriting modifications** are an extension of **archival linkage modifications** and **replay preserving modifications** that rather than relying on the rewriting of URL to be done by the archive server side, directly override JavaScript web and DOM APIs [30, 12, 18] to provide URL rewrites client side. This class of modification is an extension of **archival linkage modifications** and **replay preserving modifications**.

```

790 function init_ajax_rewrite () {
791   if (!$wbwindow.XMLHttpRequest || !$wbwindow.XMLHttpRequest.prototype ||
    ↪   !$wbwindow.XMLHttpRequest.prototype.open) return;
792   var orig = $wbwindow.XMLHttpRequest.prototype.open;
793   function open_rewritten (method, url, async, user, password) {
794     if (!this._no_rewrite) { url = rewrite_url(url); }
795     if (async !== false) { async = true; }
796     result = orig.call(this, method, url, async, user, password);
797     if (!starts_with(url, "data:")) {
798       this.setRequestHeader('X-Pywb-Requested-With', 'XMLHttpRequest');
799     }
800   }
801   $wbwindow.XMLHttpRequest.prototype.open = open_rewritten;
802   override_prop_extract($wbwindow.XMLHttpRequest.prototype, "responseURL");
803 }
804 function init_fetch_rewrite () {
805   if (!$wbwindow.fetch) { return; }
806   var orig_fetch = $wbwindow.fetch;
807   $wbwindow.fetch = function (input, init_opts) {
808     if (typeof(input) === "string") {
809       input = rewrite_url(input);
810     } else if (typeof(input) === "object" && input.url) {
811       var new_url = rewrite_url(input.url);
812       if (new_url !== input.url) {
813         input = new Request(new_url, input);
814       }
815     }
816     init_opts = init_opts || {};
817     init_opts["credentials"] = "include";
818     return orig_fetch.call(this, input, init_opts);
819   }
820 }

```

Fig. 48. Direct function overriding of JavaScript HTTP request APIs

```

2803  override_html_assign($wbwindow.HTMLElement, "innerHTML");
2804  override_html_assign($wbwindow.HTMLIFrameElement, "srcdoc");
2805  override_html_assign($wbwindow.HTMLStyleElement, "textContent");
2806  override_prop_extract($wbwindow.Document.prototype, "URL");
2807  override_prop_extract($wbwindow.Document.prototype, "documentURI");
2808  override_prop_extract($wbwindow.Node.prototype, "baseURI");
2809  override_attr_props();
2810  init_insertAdjacentHTML_override();
2811  override_iframe_content_access("contentWindow");
2812  override_iframe_content_access("contentDocument");
2813  override_func_first_arg_proxy_to_obj($wbwindow.MutationObserver, "observe");
2814  override_func_first_arg_proxy_to_obj($wbwindow.Node,
    ↪  "compareDocumentPosition");
2815  override_func_first_arg_proxy_to_obj($wbwindow.Node, "contains");
2816  override_func_first_arg_proxy_to_obj($wbwindow.Document, "createTreeWalker");
2817  override_func_this_proxy_to_obj($wbwindow, "getComputedStyle", $wbwindow);
2818  override_func_this_proxy_to_obj($wbwindow.EventTarget, "addEventListener");
2819  override_func_this_proxy_to_obj($wbwindow.EventTarget,
    ↪  "removeEventListener");
2820  override_frames_access($wbwindow);
2821  if (!wb_opts.skip_setAttribute) {
2822    init_setAttribute_override();
2823    init_getAttribute_override();
2824  }
2825  init_svg_image_overrides();
2826  init_attr_overrides();
2827  init_cookies_override();
2828  init_createElementNS_fix();
2829  if (!wb_opts.skip_dom) { init_dom_override();}
2830  init_registerPH_override();
2831  init_beacon_override();
2832 }
2833 init_document_obj_proxy($wbwindow.document);

```

Fig. 49. Direct element attribute and content rewriting

4.4 NON-SANDBOXING REPLAY

Non-sandboxing replay is the style of replay that does not separate the replayed memento from the archive-controlled portion of replay. Unlike sandboxing replay, the contents of the replayed memento and the archive added portion of 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 sandboxing replay <http://2016.makemepulse.com/> on 2017-10-22T01:59:01Z when replayed from the Internet Archive (Figure 50).

```

<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 src="/web/20171022015901js_/http://2016.makemepulse.com/javascripts/app-29f28c8...js"></script>
  </body>
</html>

```

Fig. 50. Example of non-sandboxing replay. The replayed page's contents are outlined in red. <https://web.archive.org/web/20171022015901/http://2016.makemepulse.com/>

Figure 50 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 52 (green box), which is displayed over the contents of the replayed memento (red box). To better illustrate how the archive-injected assets exist alongside the replayed mementos consider Figure 51, which displays the frame tree for the replayed memento. The frame tree for non-sandboxing replay (Figure 51) unlike the frame tree associated with sandboxing replay (Figure 42), consists of a single frame, the top frame.

The top frame for non-sandboxing replay acts as both the frame containing the non-replayed archived added portion of replay and the frame where replay actually occurs (as is the case in sandboxing replay). The non-replayed archived added portion (green boxes) are not isolated from the replayed memento (red boxes) as seen in Figure 51. This leaves the archive-injected markup vulnerable to either direct or indirect tampering with by the replayed memento simply because they exist in the same frame (origin), web.archive.org. For example, the CSS definitions contained within mementos of <http://example.com> when replayed from the Internet Archive expose the existence of the injected banner's container `div` (Figure 54).

The markup for the Internet Archive's injected banners, seen in the green box of Figure 55, consists of a wrapper `div` and the markup for the banner itself. The wrapper `div` is used by the Internet Archive to allow the banner to fill the width of the page even though they are fixing the actual banner to the very top of the page. The wrapper `div` also includes the inline style definition "display:block", giving the wrapper only the width necessary to "float" the banner to its fixed top position and have it flow naturally with the page. Because the mementos of <http://example.com> contain a style definition applied to all `div` elements contained in the page, red box in Figure 55, the wrapper `div` becomes visible by shifting the contents of mementos, downwards as highlighted by the red box of Figure 54. Due to non-sandboxing replay's lack of isolation of the necessarily archived controlled portion of replay (banner) from the replayed mementos, the mementos of <http://example.com> do not replay as they existed on the live web (Figure 53).

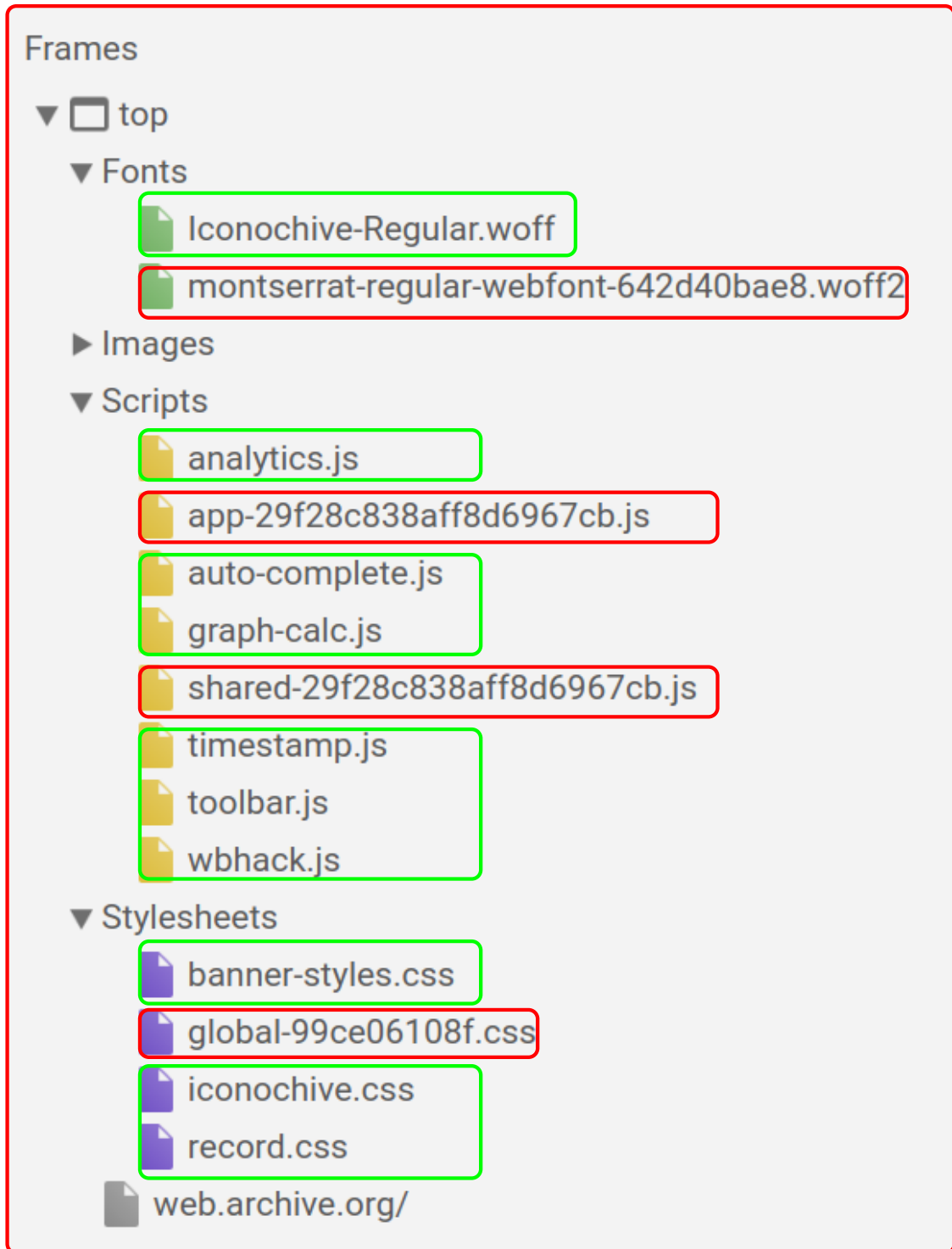


Fig. 51. Non-Sandboxing replay frame tree. <https://web.archive.org/web/20171022015901/http://2016.makemepulse.com/>

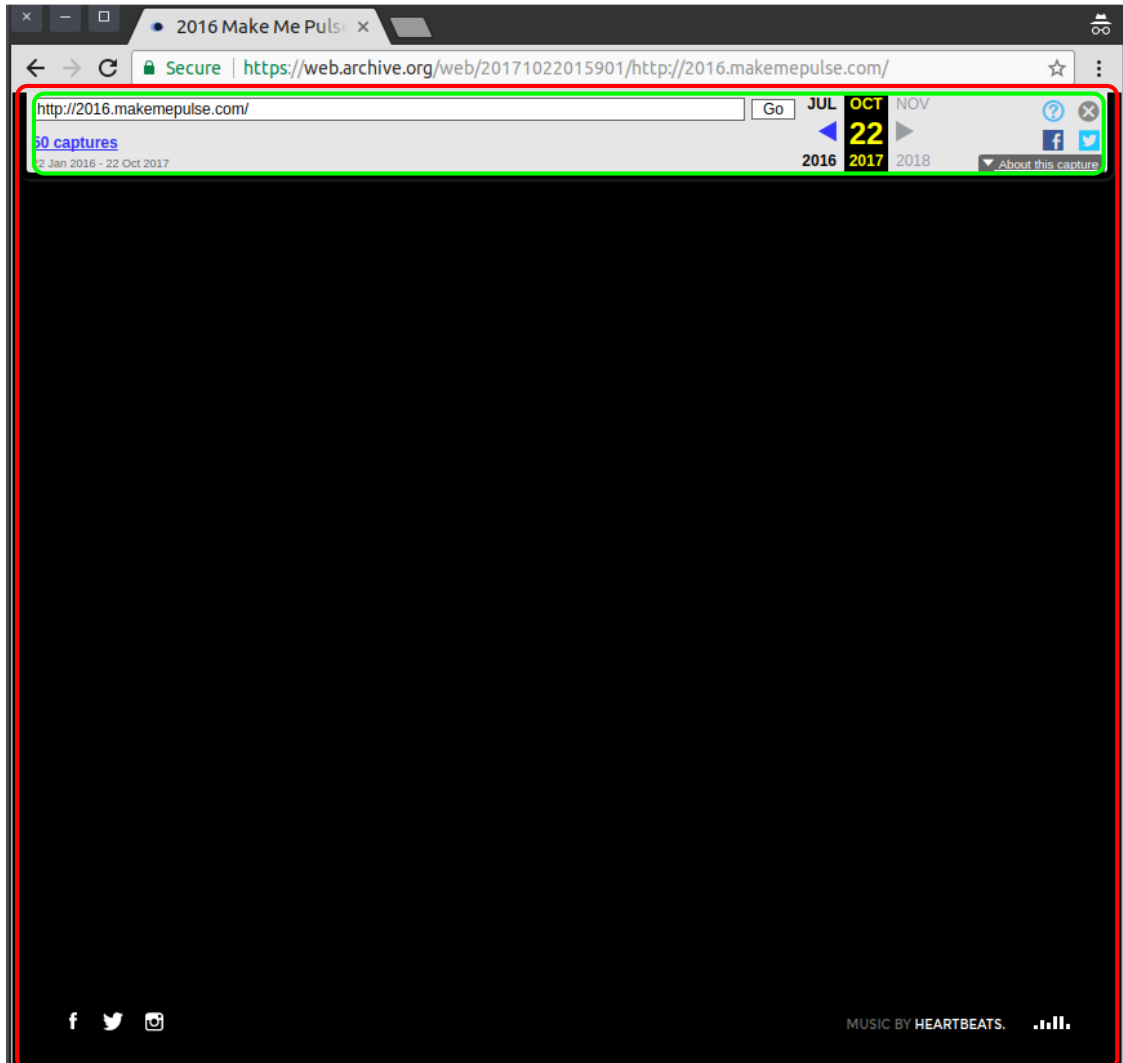


Fig. 52. Non-Sandboxing replay user view. The green box represents the necessarily archive control portion of replay and the red box represents the replayed page

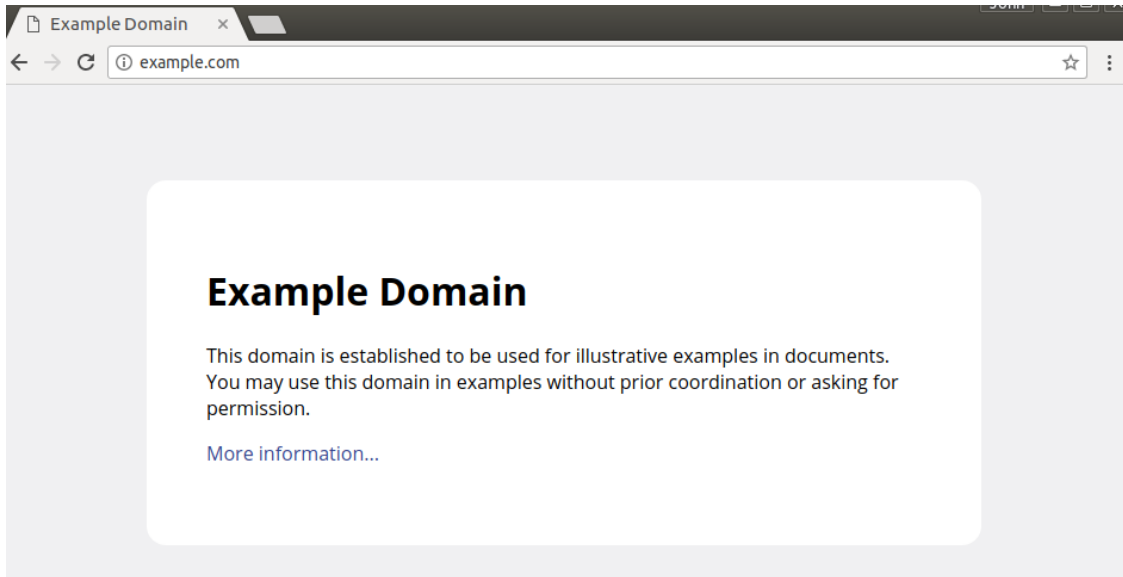


Fig. 53. `http://example.com` as it exists on the live web

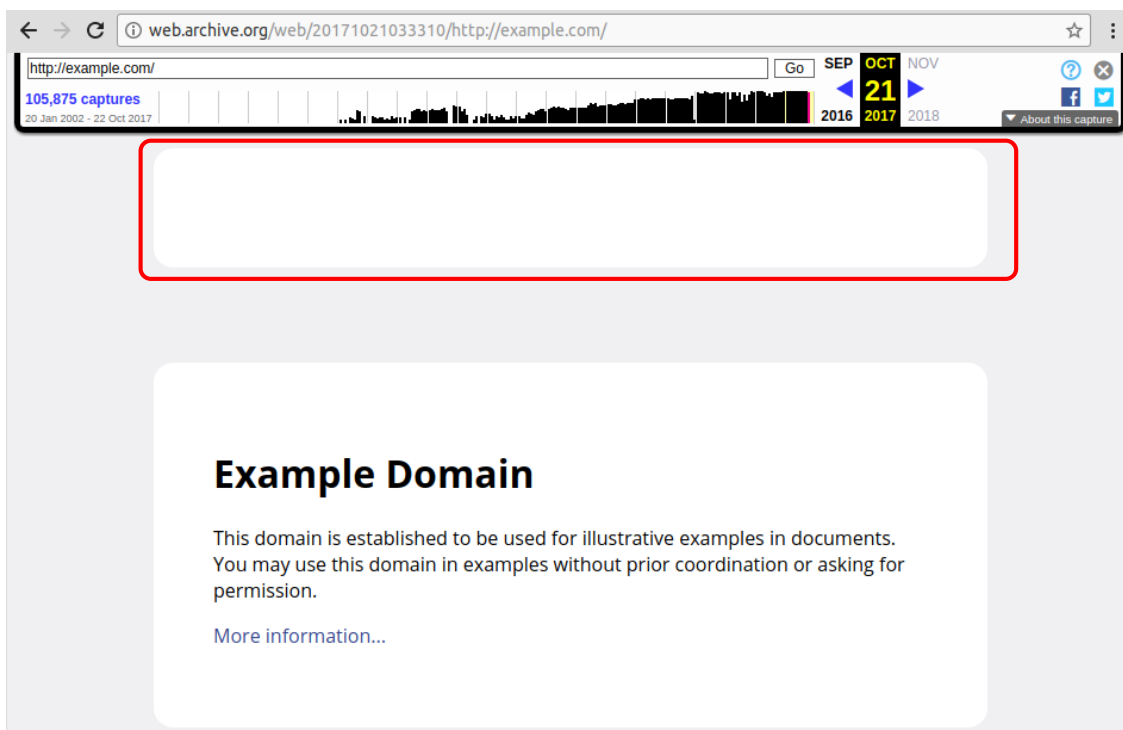


Fig. 54. Internet Archive's injected banners vulnerability to a memento embedded CSS due to non-sandboxing replay

```

▼<style type="text/css">
  body {
    background-color: #f0f0f2;
    margin: 0;
    padding: 0;
    font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
  }
  div {
    width: 600px;
    margin: 5em auto;
    padding: 50px;
    background-color: #fff;
    border-radius: 1em;
  }
  a:link, a:visited {
    color: #38488f;
    text-decoration: none;
  }
  @media (max-width: 700px) {
    body {
      background-color: #fff;
    }
    div {
      width: auto;
      margin: 0 auto;
      border-radius: 0;
      padding: 1em;
    }
  }
</style>
</head>
▼<body>
  ▼<div id="wm-ipp" lang="en" style="display: block; direction: ltr;">
    ▶<div style="position: fixed; left: 0; top: 0; right: 0;">...</div>
  </div>
  <!-- BEGIN WAYBACK TOOLBAR INSERT -->

```

Fig. 55. Internet Archive’s injected banners vulnerability (Figure 54), offending *div* and memento’s embedded CSS

4.5 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 [67] of the web page being created. The news homepage archiving platform, PastPages², demonstrates this (Figure 56).

²<http://www.pastpages.org>

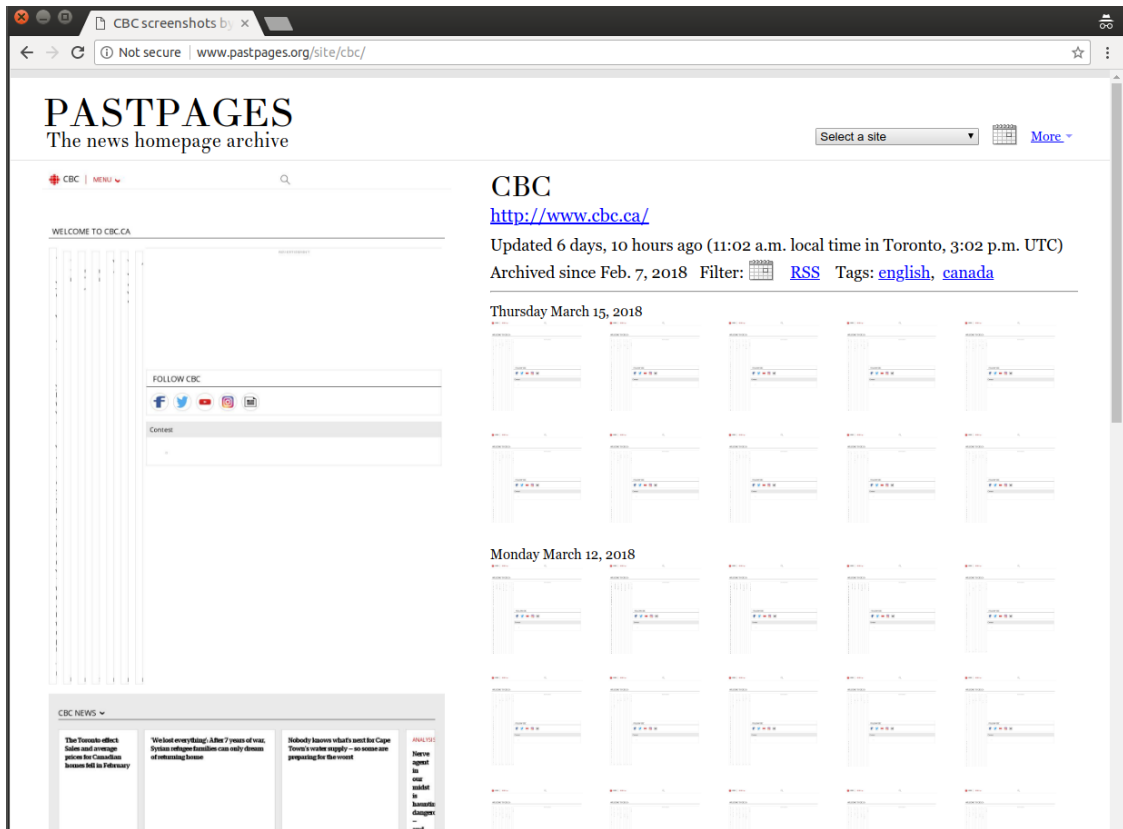


Fig. 56. PastPages preserving news sites essence through images

PastPages was created by Ben Welsh, editor of the *Los Angeles Times Data Desk*, to take screenshots of the home pages for 121 news site 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 otherwise unavailable in this archive. The images saved by this service only prove that the home pages for 121 news sites did in fact exist and what was captured in the screenshot was displayed to the tool or entity that took the screenshot, i.e. their essence. The other aspect of Essence Preservation is *Archival Caricaturization*.

4.5.1 ARCHIVAL CARICATURIZATION

Archival Caricaturization is a style of preservation that does not preserve the web page faithfully, 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”³ 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 and 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, one cannot retrieve the original web page and its embedded resources at archival time from an archive using this style of preservation.

4.5.2 ARCHIVAL CARICATURIZATION REPLAY

In order to understand *Archival Caricaturization* replay, consider the example in Figure 57 that demonstrates three different ways to make a `div` have a green background.

³<https://www.merriam-webster.com/dictionary/caricature>

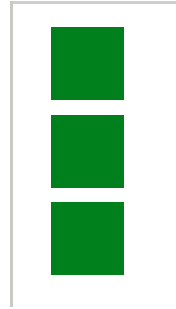
```

1 <html>
2 <head>
3   <meta name="seo" content="index me!" />
4   <style>
5     .greenBG {
6       background: green;
7       height: 25px;
8       width: 25px;
9       margin: 5px;
10    }
11    #gbg::before {
12      content: "";
13      background: green;
14      display: block;
15      height: 25px;
16      width: 25px;
17      margin: 5px;
18    }
19  </style>
20 </head>
21 <body>
22 <div style="background: green; height: 25px; width: 25px; margin:5px;"></div>
23 <div id="gbg"></div>
24 <div class="greenBG"></div>
25 </body>
26 </html>

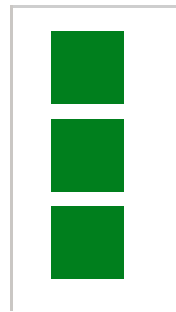
```

Fig. 57. Example <http://cs.odu.edu/~jberlin/originalThreeGreen.html>

The first `div` in Figure 57 (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 57 is shown in Figure 58a.



(a) Toy Example Rendered In Google Chrome



(b) Toy Example Archived Using Caricaturization Rendered In Google Chrome

Fig. 58. Simple example of Archival Caricaturization preserving exactly how the page existed. Rendering of HTML shown in Figure 57.

Consider the toy example after again being archived using *Archival Caricaturization* (Figure 59) and its rendering (Figure 58b). The example's original markup (Figure 57, lines 1-26) has been replaced completely and is represented by a `div` with the class `html1` (Figure 59, 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 57 lines 22) with the additional style definition “`text-align: left;`” added to it as do the remaining two green boxes (Figure 57 lines 23 and 24).

```

1 <!DOCTYPE html>
2 <html itemscope itemtype="http://schema.org/Article" prefix="og: http://ogp.me/ns# article:
  • http://ogp.me/ns/article#" style="background-color:#EEEEEE">
3 <head>
4   <!-- archive header content removed -->
5 </head>
6 <body style="margin:0;background-color:#EEEEEE">
7 <center>
8   <div id="HEADER" style="font-family:sans-serif;background-color:#FFFAE1;border-bottom:2px
  • #B40010 solid;min-width:1028px">
9     <div style="padding-top:10px"></div>
10  </div>
11  <div style="padding:10px 0;min-width:1028px;background-color:#EEEEEE"></div>
12  <div id="SOLID" style="background-color:#EEEEEE;padding-bottom:15px">
13    <div id="CONTENT" onclick=""
  • style="background-color:white;min-height:768px;max-height:100000px;position:relative;border
  • :2px #999999 solid;margin:0px -2px;width:1024px">
14      <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">
15        <div class="html" style="text-align:left;overflow-x:visible;overflow-y:visible;">
16          <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;">
17            <div style="text-align:left;background-color: green;
  • height:25px;width:25px;margin: 5px;"></div>
18            <div style="text-align:left;">
19              <span style="background-color: green;
  • display:block;height:25px;width:25px;margin: 5px;"></span>
20            </div>
21            <div style="text-align:left;background-color: green;
  • height:25px;width:25px;margin: 5px;"></div>
22          </div>
23        </div>
24      </div>
25      <div style="padding:20px 0;min-width:1028px;background-color:#EEEEEE"></div>
26    </div>
27  </center>
28 </body>
29 </html>

```

Fig. 59. Transformation of HTML shown in Figure 57 as archived through caricaturization. <http://archive.is/tOT8m>

To further illustrate *Archival Caricaturization*, consider Figure 60, with an annotated comparison of an article from <http://nocleansinging.com> on 2017-05-15 to an archived copy of the very same article as preserved by archive.is. The orange “1” annotation with arrows points out two identifiers for *Essence Preservation*, the first is *Identity Masking*. *Identity Masking* refers to the way an archive chooses to identify the web page and its dependent resources post-archival. Archives using Identity Masking change the filenames of the web page and its dependent resources to a hash of its contents or a hash seeded by some aspect of the archival process that uniquely

identifies the web page. The URI-M of the memento⁴ when replayed in archive.is was transformed into <http://archive.is/ZV75b>. The second identifier is that the page has been condensed style wise to fit into the presentational format of archive.is, an *Archival Caricaturization*. The yellow “2” annotation with arrows points out a third identifier of *Essence Preservation*, which is lack of JavaScript preservation. The search box included with the page as seen in the live web version has the JavaScript added text “Powered By Google”, whereas the archive.is memento does not because it did not preserve the page’s JavaScript. Thus, it cannot be executed on replay.

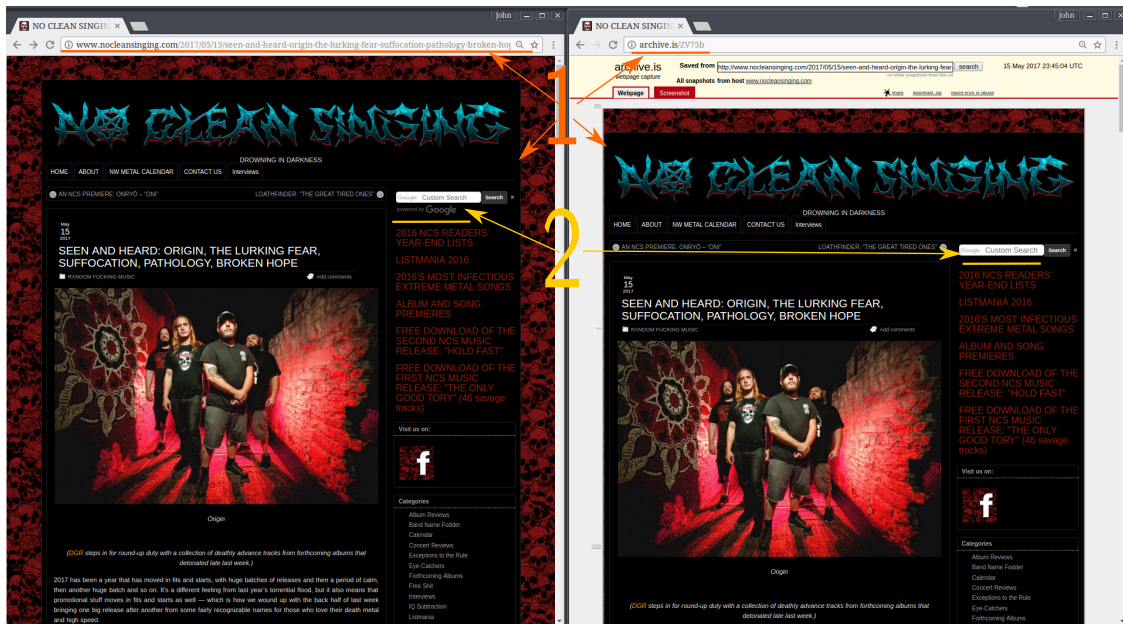


Fig. 60. archive.is caricaturization of nocleansinging.com, the live web version is on the left, and the memento is on the right

The *Archival Caricaturization* done by archive.is can also be seen in further detail when considering the URL of <http://heavyblogisheavy.com> as preserved by archive.is (Figures 61, 62, 63 and 64).

The “1” annotation in Figure 61 shows the lack of JavaScript execution and preservation, likewise for Figure 62 annotations “2” and “4”. The “2” annotation in Figure 61 shows how archive.is’ inlining of the original page’s stylesheets did not preserve the visual aesthetics of text even when no zoom has been applied, but the browser’s window takes up only 50% of the total screen size.

⁴<http://archive.today/2017.05.15-234504/http://www.nocleansinging.com/2017/05/15/seen-and-heard-origin-the-lurking-fear-suffocation-pathology-broken-hope/>

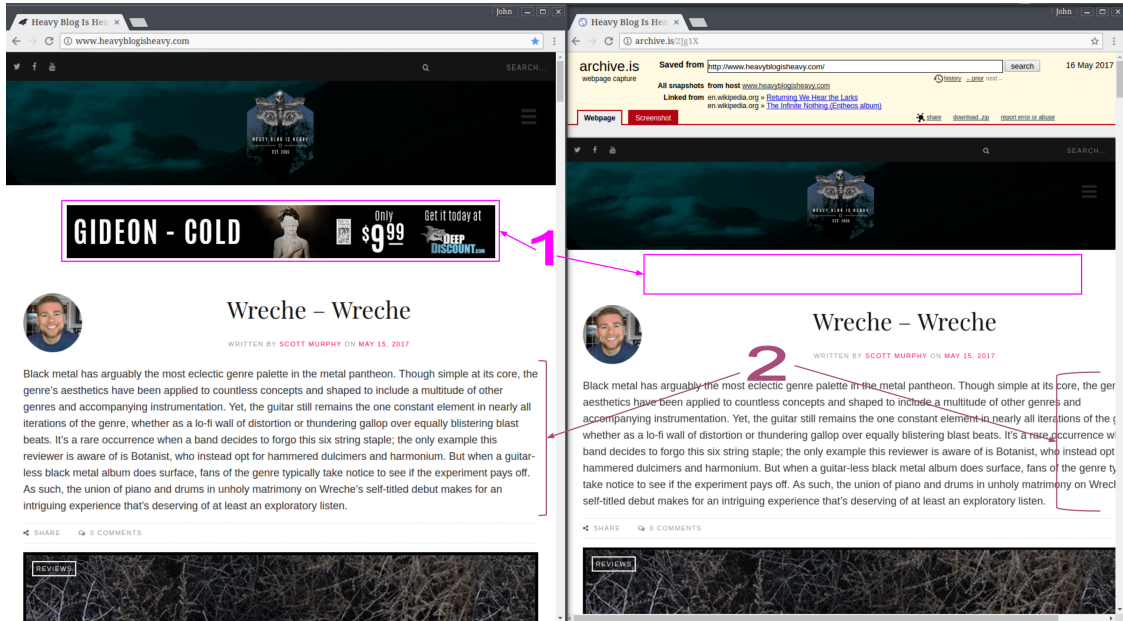


Fig. 61. heavyblogsisheavy.com vs. archive.is no zoom

Figure 62 shows the same comparison but at zoom level 50%. At this zoom level we can see another effect of the *Archival Caricaturization* applied to the page by archive.is, which is that the nav bar of heavyblogsisheavy.com is not displayed, shown in annotation “1”. The text and image placement did not reflow when zoomed out, shown in annotation “3”. Finally, there is another instance of loss of content from the original page. This is due to the lack of JavaScript archival, with the page missing the post tags highlighted by annotation “4”. Even at zoom level 90% (Figure 63), we see the same behavior as in Figure 62. The navigation bar is still not displayed, shown in annotation “1” and the text plus images do not reflow based on zoom level, shown in annotations “3” and “4”.

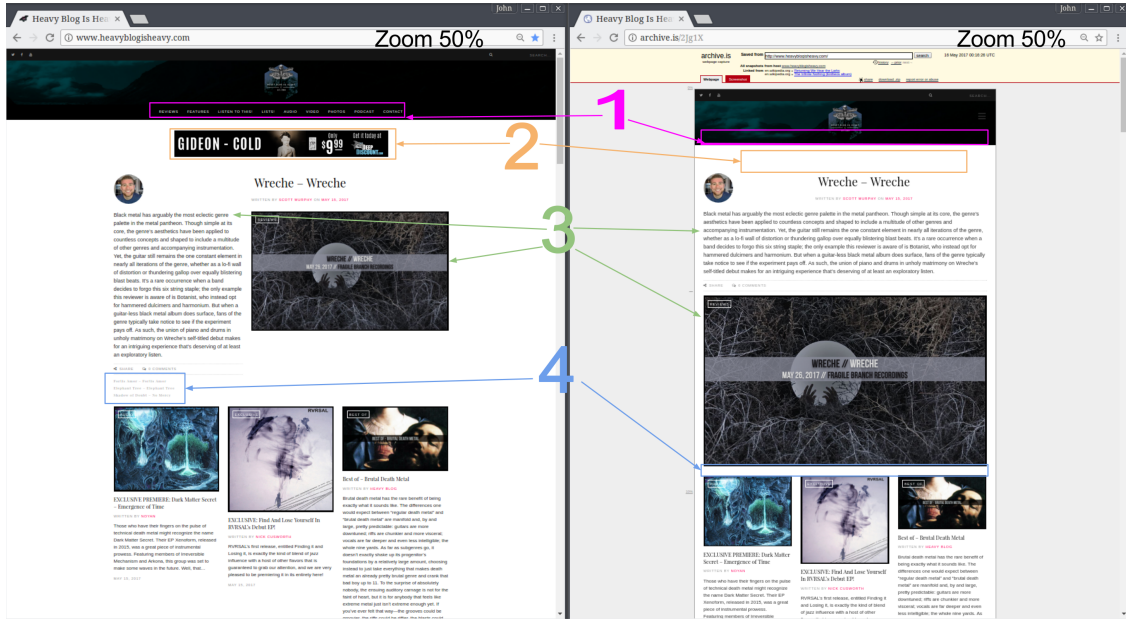


Fig. 62. heavyblogisheavy.com vs. archive.is zoom 50%

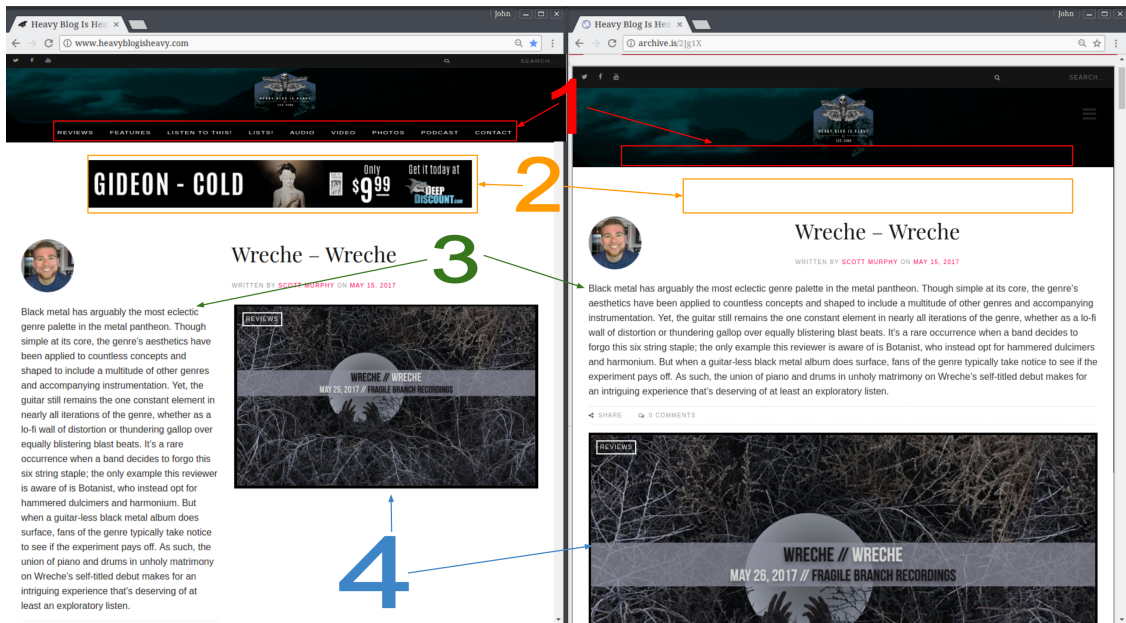


Fig. 63. heavyblogisheavy.com vs. archive.is zoom 90%

Figure 64 demonstrates the derivative transformation applied to the original resources by archive.is, which is another indication of *Archival Caricaturization* and *Essence Preservation*. It also shows what happened to the nav bar of heavyblo-gisheavy.com. The left half of Figure 64 is the page on the live web with the developer tools open, showing the class names applied to original markup. The right half of Figure 64 shows the translation of the original resource by archive.is. As shown archive.is added additional tags to the original markup to account for the “pseudo-elements” that are added through CSS only (the orange and dark blue arrows) and stripped away all original identifying markup. The red box shows the added “**display: none**” CSS style to the transformation of the header element. Even though the derivative transformation of the original markup preserved the essence of the page, it negatively impacted an essential part of the page.

An even more simple example of this can be seen when comparing⁵ (Figure 65) to a memento of the page on archive.is⁶ (Figure 66). The live web page uses Custom Elements, which are elements that are user defined only, powered through JavaScript, and have no style applied to them. As demonstrated earlier (Figure 60), archive.is neither preserves the JavaScript of the page nor does it execute the JavaScript on replay. But when replayed from archive.is (Figure 66) we see a clear view into the process for the derivative markup transformation applied by archive.is. As seen in Figure 66, archive.is did not know how to remove and replace the custom elements used by the original page, so they were left in. We also see here that archive.is does indeed add style definitions to tags that did not exist in the original, with each custom tag having the attribute style with value “**text-align:left;**” added. This style definition existed nowhere in the original as shown in Figure 65, thus further demonstrating that archive.is employs *Archival Caricaturization*.

⁵<http://wsdl-docker.cs.odu.edu:8080/tests/acidv1CustomElements>

⁶<http://archive.is/yCsqy>



Fig. 64. archive.is addition of `display:none` to heavyblogsisheavy.com


← → ↻ ⓘ wsdl-docker.cs.odu.edu:8080/tests/acidv1CustomElements/


Check [Custom Elements MDN](#) to see if your browser is supported


The Archival Acid Test

This page was setup to test the capabilities and shortcomings of archival web crawlers. More information about the rationale and individual tests can be found [below](#). For questions/comments contact [Mat Kelly](#).

The Tests

The Basics (6 tests)


Javascript (8 tests)


Advanced Features Tests (4 tests)


More Information

The Motivation
 The purpose of this web page is to test the capability of web crawlers intended for archiving (e.g., Heritrix) and potentially their corresponding replay systems (e.g., Wayback).

Tests' Rationales
 Tell an archival crawler to capture this page. Replay the capture in an archival replay system. Any non-blue squares means that some aesthetic or functionality capability of the page on the live web is not being preserved into the archive.

The Basics

- 1a - Local image, relative to the test*
- 1b - Local image, absolute URI*
- 1c - Remote image, absolute*
- 1d - Inline content, encoded image*
- 1e - Scheme-less resource*
- 1f - Recursively included CSS*

JavaScript

- 2a - Script, local*
- 2b - Script, remote*
- 2c - Script inline, DOM manipulation*
- 2d - Ajax image replacement of content that should be in archive*
- 2e - Ajax requests with content that should be included in the archive, test for false positive (e.g., same origin policy)*
- 2f - Code that manipulates DOM after a certain delay (test the synchronicity of the tools)*
- 2g - Code that loads content only after user interaction (tests for interaction-reliant loading of a resource)*
- 2h - Code that dynamically adds stylesheets*

HTML5 Features

- 3a - HTML5 Canvas Drawing*
- 3b - LocalStorage*
- 3c - External Webpage*
- 3d - Embedded Objects (HTML5 video)*

(a) Page using custom elements as it existed on the live web, viewed using the Google Chrome Browser as it existed on the live web. `http://wsdl-docker.cs.odu.edu:8080/tests/acidv1CustomElements`

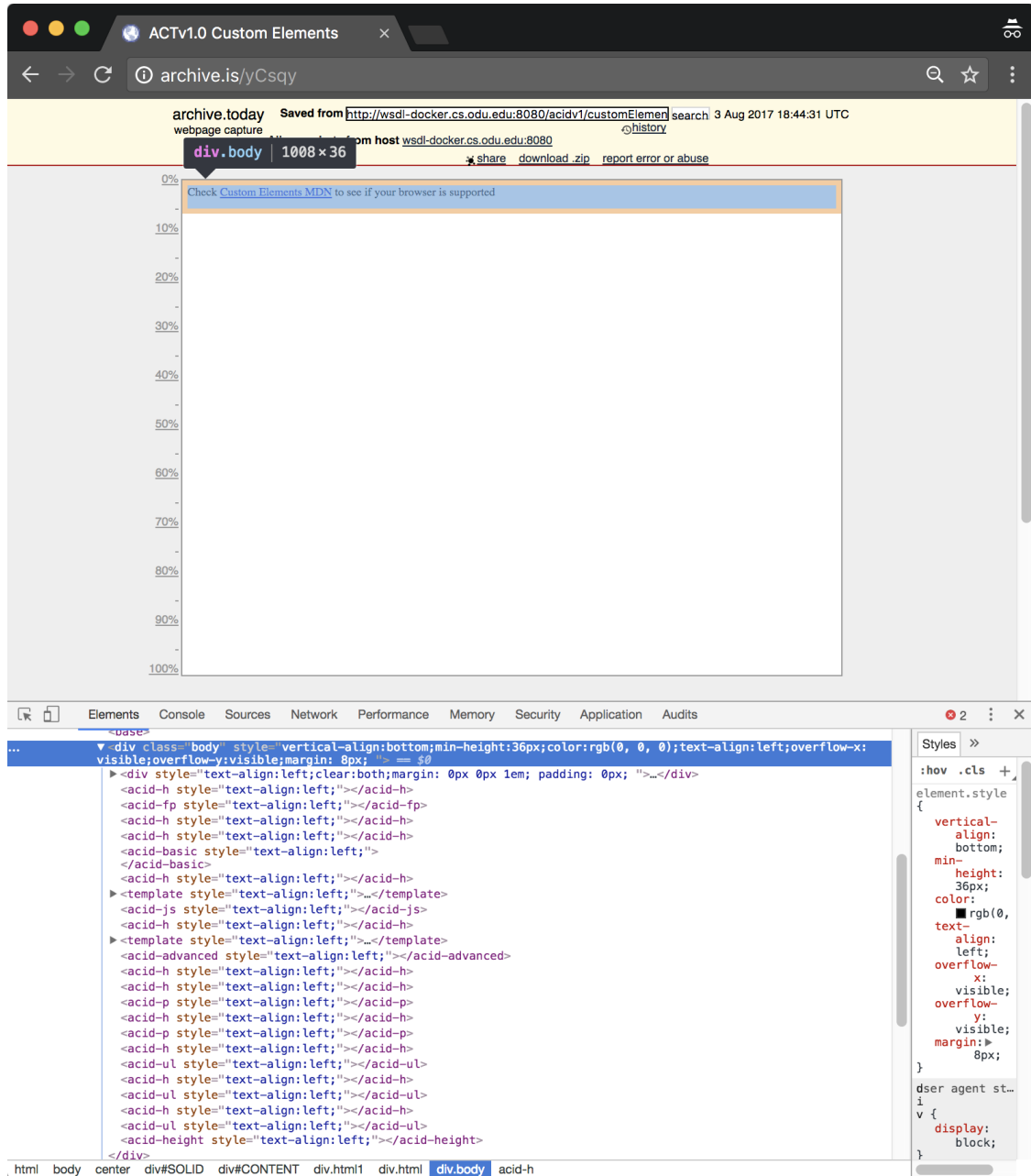


Fig. 66. Page using JavaScript powered custom HTML elements as the primary markup as replayed from archive.is demonstrating the inability of archive.is to correctly preserve the page through caricature. <http://archive.is/yCsqy>

4.6 SUMMARY

In this chapter, we classified the styles of replay and the modifications made to mementos for two archives using the “Wayback” model of replay, namely the Internet Archive and Webrecorder, and two archives using the “Non-Wayback” model, namely PastPages and archive.is.

Table 6 displays the terminology created for describing the replay styles used by the archives using the “Wayback” model. Each term listed succinctly describes the two variants of “Wayback” replay, namely **Sandboxed Replay** and **Non-Sandboxing Replay**. We showed how sandboxed replay effectively provides increased replay security in comparison to non-sandboxed replay, which is susceptible to inadvertent “attacks” by the embedded resources of the memento. This happens because non-sandboxed replay does not separate the necessarily archive controlled portion of replay (e.g., banner) from the replayed memento.

Table 6
Terminology for describing the “Wayback” model of replay

Term	Definition	Example
Sandboxed Replay	Style of replay that separates the replayed memento from the archive-controlled portion of the page through replay isolation	Webrecorder
Replay Isolation	Usage of an <code>iframe</code> to the sandbox the replayed memento, replayed from a different domain, from the archive controlled portion of replay	
Non-Sandboxing Replay	Style of replay that does not separate the replayed memento from the archive-controlled portion of replay	Internet Archive

Table 7 displays the terminology created for describing the modifications made

to mementos for facilitating replay by the “Wayback” model. Archives using the “Wayback” model of replay, must ensure **Archival Linkage** of the URI-Rs found in 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** modification in order to negate intended semantics of specific HTML element and attribute pairs to ensure that replay of the memento is possible. We showed how *Content-Security-Policy* delivering `meta` tags, used by pages on the live web to defend against malicious content injection, can prevent the archive control over replay unless the tag was modified by the archive. We then described at a high level the modifications made to the JavaScript environment of the browser by client-side rewriting called **Temporal Jailing**.

Table 7

Terminology describing the modifications made to mementos to facilitate replay using the “Wayback” model

Term	Definition
Archival Linkage	Modifications made by the archive to a page and its embedded resources in order to serve (replay) them from the archive
Replay Preserving	Modifications made on the part of an archive to negate the intended semantics of specific HTML element and attribute pairs
Temporal Jailing	The emulation of the JavaScript environment as it existed at the original memento-datetime through client-side rewriting

Lastly, we created terminology for describing both the preservation and replay of web pages using the “Non-Wayback” model (Table 8). The “Non-Wayback” model has two primary means for preservation and replaying web pages, namely **Essence Preservation** and **Archival Caricaturization**. 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 representations. The transformation process, as shown, includes **Identity Masking**, which masks the identity (URIs and filenames) of the page and its embedded resources. The radically transformed representations are then made available for replay.

Understanding the replay process and the modifications made to mementos by web archives in order to facilitate replay are important concepts required for understanding the remainder of this thesis. In the next chapter, we discuss in detail how to securely replay archived JavaScript and the modifications made to JavaScript environment of the browser by temporal jailing.

Table 8

Terminology for describing the replay of and modifications made to mementos by the “Non-Wayback” model

Term	Definition	Example
Essence Preservation	Preserves only what the web page looked like at preservation and typically results in an image, PDF, or video of web page being created	PastPages
Archival Caricaturization	The style of preservation and replay that does not preserve or replay the web page faithfully, as it originally was, but rather focuses on only preserving what it looked like at some point of time	archive.is
Identity Masking	Changes made to the URI-M of the memento and its dependent resources such that the original filenames and URI-Rs do not persist post archival but rather are transformed to an archive dependent identifier	

CHAPTER 5

JAVASCRIPT FOR THE PRESERVATION AND REPLAY OF THE MODERN WEB

In Chapter 4, we classified the modifications made to a page by an archive to facilitate replay and outlined the differing styles of replay. Also in Chapter 4 we identified that web archives relying solely on server-side rewriting miss URLs used in a manner not accounted for by the archive or involving client-side execution of JavaScript by the browser, resulting in leakage from the live web and/or the inability to replay a page due to the preserved JavaScript performing an action not permissible from within the archive. Notable failures in replay due to the lack of JavaScript overrides/rewriting are the homepage of CNN [11] and user pages found on mendeley.com [15].

The current solution for overcoming the inability of server-side rewriting to rewrite URLs used in a manner not accounted for by the archive or involving client-side execution of JavaScript is the client-side 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 is what makes it an effective addition to server-side rewriting but with a singular downside, it is a handcrafted library specifically tailored for Pywb and Webrecorder.

Other archives that wish to employ client-side rewriting must implement their own version of the library. But that would only further confirm the boutique nature of client-side rewriting libraries rather than being an aid towards providing a standard general solution for the creation of a client-side rewriting library. The current reasoning of why a general solution for creating client-side rewriting libraries does not exist is that the JavaScript web and DOM APIs provided by the browser change rapidly without a central standard from which to base the solution as is the case for server-side rewriting.

To some extent, the current assumption is correct but in reality there does exist a central standard(s) from which a general solution can be derived. Consider the following excerpts from the HTML specification [12]:

This specification uses the term document to refer to any use of HTML, . . . , as well as to fully-fledged interactive applications. The term is used to refer both to Document objects and their descendant DOM trees, and to serialized byte streams using the HTML syntax or the XML syntax, depending on context

. . .

User agents that support scripting must also be conforming implementations of the IDL fragments in this specification, as described in the Web IDL specification

The first excerpt, in short, states that even the most reactive web applications and the JavaScript interfaces for interacting with the document (DOM) are governed by the HTML specification. This is confirmed by the second excerpt that states that user agents supporting web scripting (JavaScript) must abide by the Web IDL fragments contained in the specification. From these two excerpts, we know that each browser, regardless of the underlying implementation of JavaScript, their JavaScript web and DOM APIs conform to the Web IDL fragments provided by the HTML and DOM [30] specifications. Furthermore, we know that any additional specifications linked to from the HTML and or DOM specifications, namely CSS and fetch [68], will provide Web IDL definitions for their corresponding JavaScript API if one exists. Using this knowledge we can derive a generalized and standard solution for the creation of client-side rewriting.

5.1 WEB IDL

Web Interface Design Language (Web IDL) was created by the W3C to “describe interfaces intended to be implemented in web browser”, “allow the behavior of common script objects in the web platform to be specified more readily”, and “provide how interfaces described with Web IDL correspond to constructs within ECMAScript execution environments” [18]. 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` (Figure 67).

Web IDL uses interfaces to describe how the actual JavaScript [31] 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` (Figure 67, line 1-8). The primary members of an `interface` are `attributes` (line 4), describing the exposed

state of the implementing object, and **operations** (Figure 67, line 7), which describe behaviors (methods) that can be invoked on the object [18].

```

1  [extended_attributes]
2  interface identifier : identifier_of_inherited_interface {
3    [extended_attributes]
4    /* special_keywords */ attribute type identifier;
5
6    [extended_attributes]
7    /* special_keywords */ return_type identifier([extended_attributes] type identifier);
8  };
9
10 interface_identifier implements identifier_of_implemented_interface;
11
12 partial interface identifier_of_existing_interface { /* interface_members */ };
13
14 callback interface identifier { /* interface_members */ };
15
16 typedef type identifier;
17
18 enum identifier { "value 1", "...", "value n" };
19
20 dictionary identifier { type identifier = "value"; };
21
22 callback identifier = return_type (/* arguments... */);

```

Fig. 67. Web IDL syntax

The extended attributes of an **interface**, **attribute**, **operation**, or an argument of an **operation** are annotations used to describe how language bindings will handle those constructs (Figure 67, lines 1, 3, and 6-7). Extended attributes can indicate where the interface is exposed, if it has a named constructor or if it cannot be redefined (**Unforgeable**) (Figure 68, lines 1, 4). Special keywords are used to denote, for example if an **attribute** is read-only or if an **operation** is used as the underlying objects stringifier (Figure 67, lines 4 and 7).

```

1  [Exposed=Window,NamedConstructor=Audio(DOMString src)]
2  interface HTMLAudioElement : HTMLMediaElement {};
3
4  [Unforgeable]
5  interface Location {};
6
7  typedef (Request or USVString) RequestInfo;

```

Fig. 68. Web IDL extended attributes and typedefs

Web IDL expresses the composition of an **interface** using **inheritance**, **implements**, and **partial interfaces** (Figure 67, lines 2, 10, and 12). If an **interface** inherits from or implements another **interface**, the inheriting or implementing **interface** also contain the members of the inherited or implemented **interface**. Whereas **partial interfaces** (Figure 67, line 12) are used as an “editorial aid”, allowing the definition of an existing **interface** to be separated from the main definition but are considered members of the original **interface** [18]. **Callback interfaces**, on the other hand (Figure 67 line 14), are used to describe the shape of a user created argument supplied to an **operation** of an **interface**.

Typedefs in Web IDL are used to declare a new name for a type or a union of many types (Figure 67, line 16 and Figure 68, line 7) and are only a naming shorthand for referencing the type(s). The remaining constructs of Web IDL being used to express a type for valid predefined strings are an **enum**, fixed key value pairs, a **dictionary**, and a **callback**, which is a named function type (Figure 67, lines 18-22).

5.2 WEB IDL JAVASCRIPT MAPPING

Each of the constructs defined by Web IDL have their own mapping to the JavaScript execution environment. For the purpose of this thesis we will only go into detail into how the Web IDL **interface** type maps to the JavaScript execution environment and any additional information necessary for the creation of a client-side rewriting library based on a supplied set of Web IDL definitions (fragments). The Web IDL specification states that each for 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 [18].

Interfaces which are not **callback** interfaces or declared with the **NoInterfaceObject** extended attribute (Figure 69, line 8) are represented by an **interface** object exposed on the global environment object as a named property and have a **prototype object** [31, 18]. The name of the property is the **interface**’s identifier with a value consisting of its static and constant members called an **interface object** that is both *writable* and *configurable* on the global object unless it was defined as an attribute of the global object with the **Unforgeable** extended attribute or is the global object itself [18] (Figure 69 lines 1,4-5).

```

1  [Unforgeable]
2  interface Location {};
3
4  interface Window : EventTarget {
5    [Unforgeable] readonly attribute Document document;
6  };
7
8  [NoInterfaceObject]
9  interface URLUtils {};

```

global object

Fig. 69. Unforgeable and NoInterfaceObject extended attributes

Unlike the restrictions placed on the exposed `interface` objects, a `prototype` object exists for each implemented `interface` and is exposed on the global environment object regardless of if the `interface` was defined with the `NoInterfaceObject` extended attribute. Each `attribute` and `operation` exists on the `prototype` object as a named property whose name is the identifier of the `attribute` or `operation` with the following additional properties. `Attributes` are defined to be *configurable* unless the `attribute` was defined with the `Unforagable` extend attribute and have a *getter* and *setter* function. Likewise, `operations` are defined to be *configurable* unless the `operation` was defined with the `Unforagable` extend attribute except they do not have a *getter* or *setter*. In order to avoid confusion when speaking about `interface` objects and `prototype` objects, consider the representation for HTML anchor tags in both JavaScript (Figure 70) and Web IDL (Figure 71).

When the Web IDL specification states that an interface has both an `interface` object and an `interface` `prototype` object, it is referring to lines 1 and 2 of Figure 70. The `interface` object exists on the global JavaScript environment object (`window`) as a named property, its IDL interface identifier (Figure 70, line 1). The `prototype` object for the `interface` exists as the `prototype` property on the `interface` object (Figure 70, line 4). When you create a new `HTMLAnchorElement` (Figure 70, 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 70, 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. 70. Web IDL interface to ECMAScript mapping

Now consider the actual Web IDL fragment for the `HTMLAnchorElement`, seen with annotations in Figure 71. 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` was not defined with the `NoInterfaceObject` extended attribute and has both. Because `HTMLAnchorElement` implements `URLUtils`, the attributes and operations on the `URLUtils` `prototype object` are also on `HTMLAnchorElements`. This is the same for the `HTMLElements` attribute and operations, as it is extended (inherited) by `HTMLAnchorElement`.


```

1 interface HTMLAnchorElement : HTMLInputElement ← Inherited prototype object
2   [Reflect] attribute DOMString target;
3   [Reflect] attribute DOMString download;
4   [Reflect] attribute DOMString ping;
5   [Reflect] attribute DOMString rel;
6   [Reflect] attribute DOMString hreflang;
7   [Reflect] attribute DOMString type;
8   [Reflect] attribute DOMString referrerpolicy; } Prototype object of
9   [Reflect] attribute DOMString text;           the interface object.
10  [Reflect] attribute DOMString coords;
11  [Reflect] attribute DOMString charset;
12  [Reflect] attribute DOMString name;
13  [Reflect] attribute DOMString rev;
14  [Reflect] attribute DOMString shape;
15 };
16 HTMLAnchorElement implements URLUtils;
17                                     ← Implemented interface's
18 [NoInterfaceObject]                 prototype inherited
19 interface URLUtils ←
20   attribute USVString href;
21   [NotEnumerable, ImplementedAs=href] USVString toString();
22   readonly attribute USVString origin;
23   attribute USVString protocol;
24   attribute USVString username;
25   attribute USVString password;
26   attribute USVString host;
27   attribute USVString hostname;
28   attribute USVString port;
29   attribute USVString pathname;
30   attribute USVString search;
31   attribute USVString hash;
32 };

```

Fig. 71. HTMLAnchorElement.idl

5.3 AUTO-GENERATING A CLIENT-SIDE REWRITER

The fundamental goal of URL rewriting is to ensure that every URL found in or operated on by the archived resources 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. Albeit, server-side performs additional rewriting beyond those well-known locations to account for the capabilities of JavaScript. But JavaScript can only retrieve the value

of arbitrary attributes, introduce additional HTML into the document, modify the existing HTML of the document, and make HTTP requests using the APIs described by the Web IDL fragments included or linked to by the specifications that are the basis for HTML and CSS rewriting. It is from those Web IDL fragments in combination with the description of how Web IDL maps to the JavaScript environment that the auto-generation of a client-side rewriting library is possible.

5.3.1 IDENTIFYING WEB IDL INTERFACES

Fundamentally both client-side and sever-side rewriting operate under the same constraints, that is, both cannot rewrite URLs unless they know where to look. Server-side rewriting looks for URLs in HTML based off the tag and attribute name (Table 9). Similarly, rewriting CSS looks for URLs contained in the `import` or `url` keywords (Figure 72). 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 “identifiers” used by server-side HTML and CSS rewriting.

Table 9
HTML Element Attributes With Rewrite Modifier From Pywb

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

As seen in Table 9, server-side rewriting of HTML identifies a tag to be rewritten using the tag and attribute name. Note that the rewrite modifiers displayed in Table 9 differ from the modifiers used by OpenWayback (Figure 26). We chose to use the rewrite modifiers from Pywb because they follow the convention set by the Internet Archive's Wayback Machine. But as shown by Figures 68 and 71, the naming conventions for HTML element interfaces in Web IDL do not match the actual tag name. Rather, the match is found when considering the identifiers of attributes that are the same as the attribute names used by server-side rewriting. This provides

us with a total of seven well-known URL identifiers (namely: *action*, *content*, *data*, *href*, *poster*, *src*, *srcset*) for identifying the HTML element interfaces as described in Web IDL. The *style* attribute, the eighth identifier, unlike the other seven attributes, also doubles as an HTML element whose text content contains the full set of allowed CSS style definitions. Because the *style* attribute also doubles as a tag, we know two things: 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 (Figure 72).

Each of the style definitions seen within the *style* attribute (Figure 72, lines 1-5) are considered properties of a CSS declaration block that is exposed as the Web IDL interface `CSSStyleDeclaration` with two ways of mutating the state of the attribute [69, 31, 18, 30]. The first mutation is when the attribute is set directly, i.e. `elem.style = '...'`, causing the plain text string to be converted into a `CSSStyleDeclaration` whose properties are the style declarations contained within the string. The conversion algorithm used for the first mutation is the same one used when mutating the text contents of a `style` tag (Figure 72, lines 6-10). The second means of mutation is by direct modification of a property of the `CSSStyleDeclarations` via one of its identifiers, i.e. `elem.style.width = '....'`.

```

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

```

Fig. 72. CSS style properties that may contain URLs and how URLs may exist in CSS style definitions found in a style tag

Borrowing once more from server-side rewriting, we know that when rewriting CSS, the URLs to be rewritten can only be found in `import` and `url` keywords and, when considering Figure 72, we find the totality of identifiers for client-side rewriting of CSS. Each of the five elements with the *style* attribute found in Figure 72 contain one of the five `CSSStyleDeclaration` attribute identifiers whose value may contain a

URL but with a minor caveat. The caveat for the `CSSStyleDeclaration` attributes is when accessed directly via the `style` attribute the property names are camel-case but attributes may be set or retrieved using both camel-case and actual style declaration name via the APIs provided by the `CSSStyleDeclaration` (Figure 73) [69].

Also, found in Figure 73 are two additional identifiers for the CSS rewriting component, `setProperty` and `cssText` (lines 2 and 3). The Web IDL interface for the `style` tag, `HTMLStyleElement`, does not contain a text content attribute implying, it must be found on an inherited or implemented interface.

```

1 interface CSSStyleDeclaration {
2   void setProperty(DOMString property, DOMString value, optional DOMString priority);
3   attribute DOMString cssText;
4   getter (DOMString or float) (DOMString name);
5   getter DOMString item(unsigned long index);
6   setter void (DOMString property, DOMString? propertyValue);
7 };
8
9 interface HTMLStyleElement : HTMLInputElement {};
10
11 interface HTMLInputElement : Element {
12   attribute CSSStyleDeclaration style;
13 };
14   ↑
15   Every HTMLInputElement can have a style attribute
16
17 interface Element : Node {};
18
19 interface Node : EventTarget {
20   attribute DOMString? textContent;
21 };

```

Inheritance Hierarchy

```

Node
  ↑
Element
  ↑
HTMLInputElement
  ↑
HTMLStyleElement

```

Can only access or mutate URLs in style definitions from HTMLStyleElement

Fig. 73. CSS in Web IDL

That is indeed the case when considering the inheritance hierarchy of the interface representing the `style` tag, `HTMLStyleElement`. Hierarchy annotated to the right of the interfaces is shown in Figure 73 lines 9-19. Because the `Node` interface exposes the text contents of every `Node` as the `textContent` attribute and the `HTMLStyleElement`'s inheritance hierarchy includes `Node` as the topmost parent interface, we know that the `textContent` attribute is also an attribute of `HTMLStyleElement`. Likewise, we know that every interface inheriting from `HTMLInputElement` has a `style` attribute that is a `CSSStyleDeclaration`. Since the `textContent` attribute of all `Node` inheriting interfaces can only be used to introduce potentially un-rewriting URLs using the attribute on `HTMLStyleElement`'s prototype object, we need to only override the access and modification of it on the `HTMLStyleElement`. To demonstrate this, consider the page <http://www.cs.odu.edu/~jberlin/simpleStyle.html>, seen in Figure 74

which provides viewers of the page the ability to dynamically change the *textContent* attribute of an existing `style` tag.

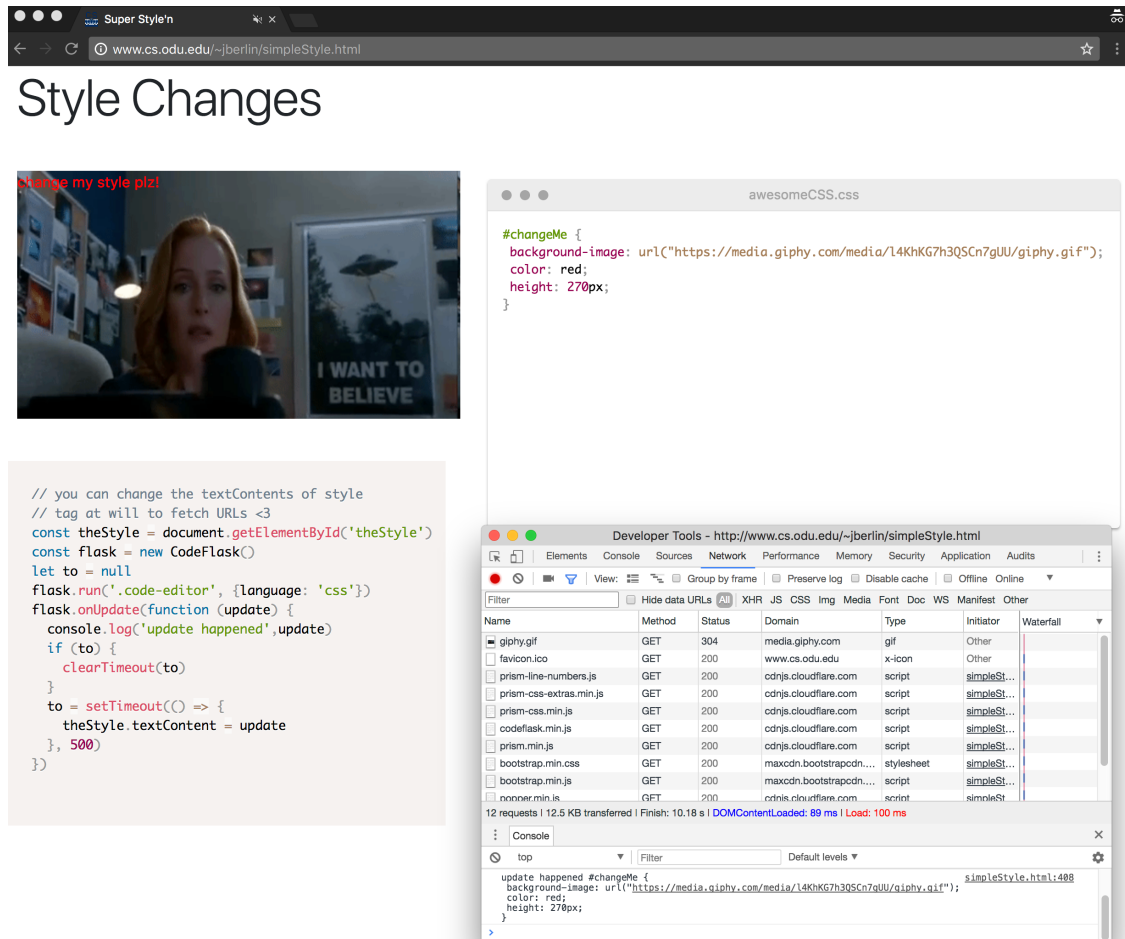


Fig. 74. Dynamic changes to the *textContent* attribute of a `style` tag to load URLs live web. <http://www.cs.odu.edu/~jberlin/simpleStyle.html>

The page provides its viewers with the ability to edit a fake css file named `awesomeCSS.css` via a faux code editor to dynamically change the style of an element. When a viewer changes the text contents of the editor, which is a `textarea` element [12], a small bit of JavaScript (seen below the styled element) takes its value and sets the *textContent* attribute of a `style` tag. As seen in Figure 74, when the *background-image* style definition (Figure 72, line 1) was added with the value `url("https://media.giphy.com/media/14KhKG7h3QSCn7gUU/giphy.gif");` and the JavaScript updated the *textContent* attribute of the `style` tag (`theStyle`), the browser initiated a resource fetch and the gif was displayed as the background of the `#changeMe` element. But the behavior of the page does not persist when

archived using the save page now feature of the Wayback Machine and then replayed (Figure 75). This is to be expected as the Wayback Machine uses a content security policy to block un-rewritten URLs from being loaded by the page. However, this is not the expected behavior for pages which access and mutate the *style* attribute or any attribute of HTML elements that can be used to initiate browser resource fetches using an identifier(s) from the next set of well-known identifiers (Figure 76).

The screenshot shows a browser window with the URL `http://web.archive.org/web/20180224044958/http://www.cs.odu.edu/~jberlin/simpleStyle.html`. The page content includes the text "change my style plz!". A code editor displays the following JavaScript code:

```
// you can change the textContents of style
// tag at will to fetch URLs <3
const theStyle = document.getElementById('theStyle')
const flask = new CodeFlask()
let to = null
flask.run('.code-editor', {language: 'css'})
flask.onUpdate(function (update) {
  console.log('update happened', update)
  if (to) {
    clearTimeout(to)
  }
  to = setTimeout(() => {
    theStyle.textContent = update
  }, 500)
})
```

The developer tools console shows a network error:

```
Refused to load the image 'https://media.giphy.com/media/14KhKG7h3Q5Cn7gUU/giphy.gif' because it violates the following Content Security Policy directive: 'default-src: self'...
```

Fig. 75. Dynamic changes to the *textContent* attribute of a *style* tag to load URLs archived. `http://web.archive.org/web/20180224044958/http://www.cs.odu.edu/~jberlin/simpleStyle.html`

```

1 interface Attr : Node {
2     readonly attribute DOMString name;
3     attribute DOMString value;
4     attribute DOMString nodeValue;
5 };
6
7 interface Element : Node {
8     attribute DOMString innerHTML;
9     attribute DOMString outerHTML;
10    void insertAdjacentHTML(DOMString position, DOMString text);
11    Element insertAdjacentElement(DOMString where, Element element);
12
13    DOMString? getAttribute(DOMString name);
14    void setAttribute(DOMString name, DOMString value);
15
16    Attr? getAttributeNode(DOMString name);
17    Attr? setAttributeNode(Attr attr);
18 };

```

Attributes (Attr) are Nodes

Well-known markup accessing and mutation identifiers

Well-known attribute accessing and mutation identifiers

Fig. 76. Well-known property mutating identifiers

The attributes of HTML elements are also represented as instances of the `Attr` interface (Figure 76, lines 1-5), accessible or mutable through the `getAttributeNode` and `setAttributeNode` operation identifiers of the `Element` interface (lines 16-17). But because we are only concerned with the access or mutation of attribute values when accessed from instances of the `Attr` interface, we can disregard those operation identifiers from `Element` for retrieving or setting `Attr` interfaces. This leaves only the `getAttribute` and `setAttribute` operation identifiers of the `Element` interface to be considered for retrieving or mutating attributes (Figure 76, lines 13-14). It must be noted that each case for the access and mutation of an attribute is unique, which requires handling each case individually. Returning to the `Element` interface, we find that it also exposes the identifiers `innerHTML`, `outerHTML`, and `insertAdjacentHTML`, which are used to insert document markup via strings, and the `insertAdjacentElement` identifier used to insert new element instances into the document.


```

1 interface Node : EventTarget {
2     Node insertBefore(Node node, Node? child);
3     Node appendChild(Node node);
4     Node replaceChild(Node node, Node child);
5 };
6
7 interface HTMLIFrameElement : HTMLElement {
8     attribute DOMString srcdoc;
9 };
10
11 interface Document : Node {
12     void write(DOMString... text);
13     void writeln(DOMString... text);
14 };

```

} Well-known markup
mutation identifiers

Well-known markup accessing
and mutation identifiers

Well-known markup
mutation identifiers

Fig. 77. Well-known document mutating identifiers

Likewise, the `Node` interface exposes identifiers `insertBefore`, `appendChild`, and `replaceChild`, which also mutate document markup but via instances of the `Node` interface. It is also important to note that the means for mutating the document's markup using an instance of the `Element` or `Node` interface are required because of interfaces like `DOMParser` (Figure 78) [30, 70].

```

1 [Constructor]
2 interface DOMParser {
3     [NewObject] Document parseFromString(DOMString str, SupportedType type);
4 };

```

Fig. 78. DOMParser Web IDL Interface

The `DOMParser` interface provides access to the document parser of the browser and will not be overridden in order to facilitate the rewriting of HTML elements inserted into the document as strings (Figure 76, lines 8-10 and Figure 77, lines 8, 12-13). To illustrate the importance of overriding the interfaces shown in Figures 76 and 77, consider the following two archived pages which employ the common design pattern lazy loading:

```

1 http://web.archive.org/web/20180223141745/http://www.bbc.com/news
  /world-middle-east-26116868 (Figure 79)

```

2 <https://web.archive.org/web/20170209205035/http://www.soufeel.com>
(Figure 85a)

Lazy load page 1 replays as expected (e.g., images accompanying the news story are visible) until you scroll down to bottom of the page and notice images are missing from the additional stories' footer (Figure 79). On further inspection of the missing images you will notice that the value for their `img` tag's `src` attributes are un-rewritten URLs, which 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 content security policy (Figure 80). Also, found on further inspection of the blocked `img` tags seen in Figure 79 is an indication that the `img` tags were added to the document via JavaScript by noticing the CSS classes for the images (Figure 81) and a section of the archived HTML found below where the images are now located (Figure 82).

The screenshot shows a web browser window displaying a BBC News article page. The URL in the address bar is <http://www.bbc.com/news/world-middle-east-26116868>. The page features three placeholder images for news stories, each with the BBC logo. Below the images are the following headlines and dates:

- 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)

Below the headlines is a section titled "Why you can trust BBC News". The browser's developer console is open, showing the HTML structure of the page. The following code snippet is highlighted, showing the source of the blocked image:

```


<![endif]-->

```

Fig. 79. 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>

The screenshot shows the Network tab of browser developer tools. The page URL is `http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868`. The network log shows several requests that were blocked by the Content Security Policy (CSP). The blocked requests include:

- `news-light.svg` (GET, blocked:csp, domain: static.bbci.co.uk)
- `translations?callback&locale=en-GB` (GET, blocked:csp, domain: www.bbc.co.uk)
- `latest_breaking_news?audience=US&callback=breakingNews` (GET, blocked:csp, domain: polling.bbc.co.uk)
- `_99947493_collage.jpg` (GET, blocked:csp, domain: ichef.bbci.co.uk)
- `_99934185_mediaitem99934184.jpg` (GET, blocked:csp, domain: ichef.bbci.co.uk)
- `_99946718_mediaitem99946717.jpg` (GET, blocked:csp, domain: ichef-1.bbci.co.uk)
- `ping?h=bbc.co.uk&p=bbc.co.uk%2Fnews%2Fworld-middle.....` (GET, blocked:csp, domain: ping.chartbeat.net)
- `ping?h=bbc.co.uk&p=bbc.co.uk%2Fnews%2Fworld-middle.....` (GET, blocked:csp, domain: ping.chartbeat.net)

The last request, `world-middle-east-26116868`, is a successful GET request (200) for the document from `web.archive.org`.

Fig. 80. bbc.com new story footer images blocked by the Wayback Machine’s content security policy, network tab of browser developer tools. `http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868`

`img.responsive-image__img.responsive-image__img--loading.js-image-replace`

Fig. 81. bbc.com new story footer blocked images `img` tag CSS classes

From the CSS classes of the `img` tags for the blocked images (Figure 81), we know that the `img` tags were responsive image placeholders that are to be replaced with the real image by JavaScript using the information embedded in the two `div` tags seen in Figure 82. The first `div` (`id=comp-pattern-library-7`) is used to embed a URL in its `data-post-load-url` attribute for the retrieval of the images (Figure 79) and the second `div` (`id=comp-from-other-news-sites`) embeds the loading strategy and information about those images in its `data-comp-meta` attribute. When the page loads, the page’s JavaScript makes a request for the lazy loaded images using the URL embedded in the first `div`. The response for the request is JSON containing additional information pertaining to the internals of the page and a string of HTML which is added to document in preparation for the lazy loading of the images (Figure 83). The HTML shown in Figure 83 corresponds to the image highlighted in Figure 79. The full response can be seen in Figure 146 found in Appendix A of this thesis.

```

<div id="comp-pattern-library-7" class="hidden"
data-post-load-url="/news/pattern-library-components?options%5BassetId%5D=26116868&opt
ions%5Bcontainer_class%5D=container-more-from-this-index&options%5Bdata%5D%5Bsourc
e%5D=candy_parent_index&options%5Bdata%5D%5Bsource_params%5D%5Bsection_title%5D=1&
amp;options%5Bcomponents%5D%5B0%5D%5Bname%5D=sparrow&options%5Bcomponents%5D%5B0%5
D%5Blimit%5D=3&options%5Bloading_strategy%5D=post_load&options%5Bstats%5D%5Bli
nk_location%5D=more-section-index&options%5Bstats%5D%5Bstrapline_link_location%5D=
more-from-this-index-headline&options%5Bstats%5D%5Bsection_label%5D=more-from-this
-index-section-label&options%5Basset_id%5D=world-middle-east-26116868&presente
r=pattern-library-presenter">
</div>
<div id="comp-from-other-news-sites" class="hidden"
data-comp-meta="{&quot;id&quot;:&quot;comp-from-other-news-sites&quot;,&quot;type&quot;:&quot;
ot;from-other-news-sites&quot;,&quot;handler&quot;:&quot;default&quot;,&quot;deviceGr
oups&quot;:null,&quot;opts&quot;:{&quot;assetId&quot;:&quot;26116868&quot;,&quot;condi
tions&quot;:[&quot;is_local_page&quot;],&quot;loading_strategy&quot;:&quot;post_load&q
uot;,&quot;asset_id&quot;:&quot;world-middle-east-26116868&quot;},&quot;template&quot;:
:&quot;\/component\/from-other-news-sites&quot;}">
</div>

```

Fig. 82. Embedded configuration for lazy loading the additional stories images. <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]>

<![endif]-->
</div>
</div>

```

Fig. 83. Portion of the response to the request for the additional stories images “/news/patttern-library-components?...”. <http://web.archive.org/web/20180223141745/http://www.bbc.com/news/world-middle-east-26116868>

As shown in Figure 83, the Wayback Machine is not rewriting JSON found in the response bodies of archived requests, which causes the JavaScript code responsible for the lazying loading of the images to operate on un-rewritten URLs (Figure 84). Each `div` tag matching the CSS selector “`div.js-delayed-image-load`” is replaced with a dummy `img` tag with the real image URL as the value for the tag’s `datasrc` attribute set from the `div`’s `data-src` attribute (lines 3-9). Then when the JavaScript code has determined it is time to load the real image (lines 11-20), it retrieves the URL from

either the *datasrc* or *databgsrc* attribute and sets the *src* attribute of the element if it is an image. Otherwise, the JavaScript loads the image via the *backgroundImage* property of the element's *style* attribute. Now consider the second page (Figure 85a), which unlike the first lazy loading page (Figure 79), embeds all the information necessary for lazy loading its images using JavaScript.

```

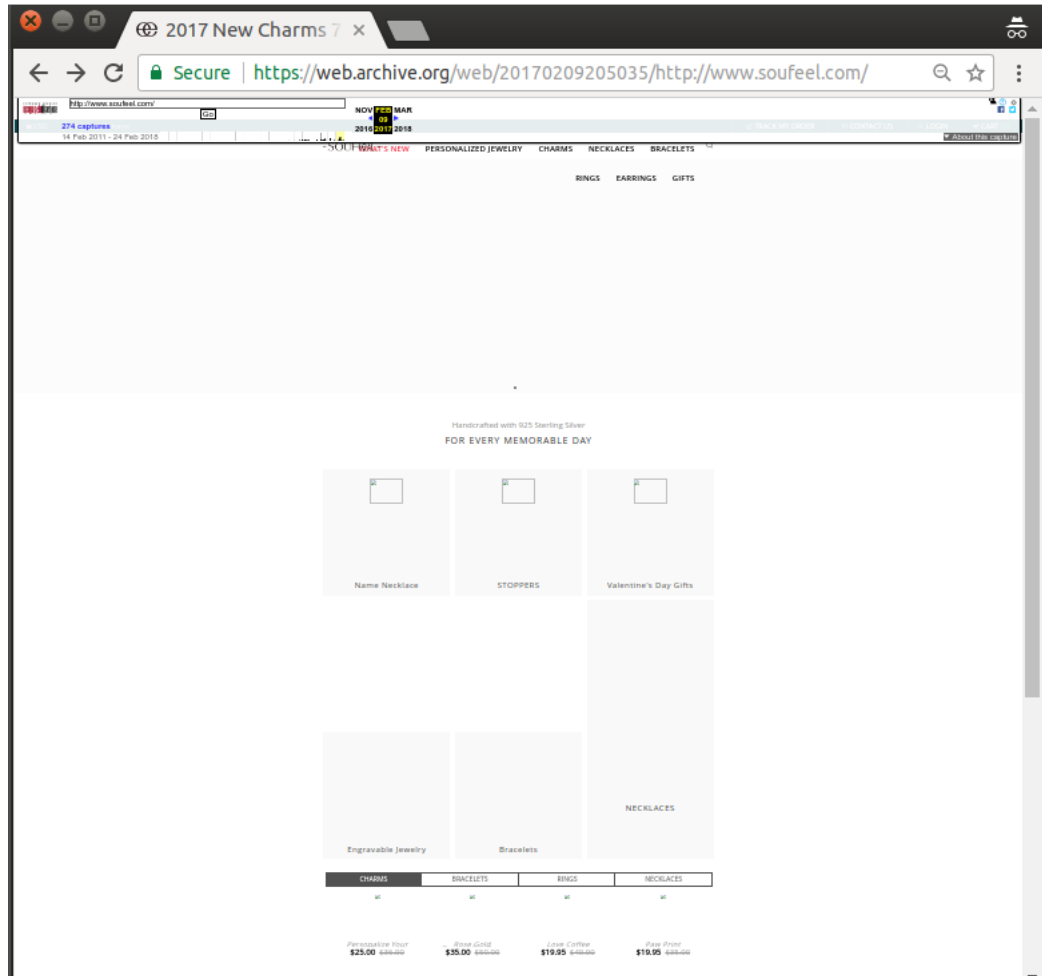
1  { // http://static.bbc.co.uk/news/1.215.02084/js/all.js
2  changeDivsToImgs: function() {
3    $("div.js-delayed-image-load").each(function(e, t) {
4      t = "number" !== typeof t ? t : e;
5      var o = t.getAttribute("data-width"), i = t.getAttribute("data-height"),
6      s = ""; Replace each lazyload div with a lazy img
7      o > 0 && i > 0 && (s = ' width="' + t.getAttribute("data-width") + ' '
8      height="' + t.getAttribute("data-height") + ' ');
9      var a = n(t.getAttribute("data-alt"));
10     $(t).replaceWith('');
15   }) set the datasrc from the data-src
16 }, attribute on img attribute from div
17 resizeImages: function(e) {
18   if (!this.isResizing) {
19     /*...code removed for presentation...*/
20     var r = this.calcImgSrc(a.getAttribute("datasrc") ||
21     a.getAttribute("databgsrc") || a.src, a.clientWidth);
22     a.src ? r && r !== a.src && a.setAttribute("src", r),
23     a.setAttribute("data-highest-encountered-width",
24     this.matchBestWidth(a.clientWidth))
25     : r && "url(" + r + ")" !== a.style.backgroundImage &&
26     (a.style.backgroundImage = "url(" + r + ")");
27     this.isResizing = !1
28   }
29 }

```

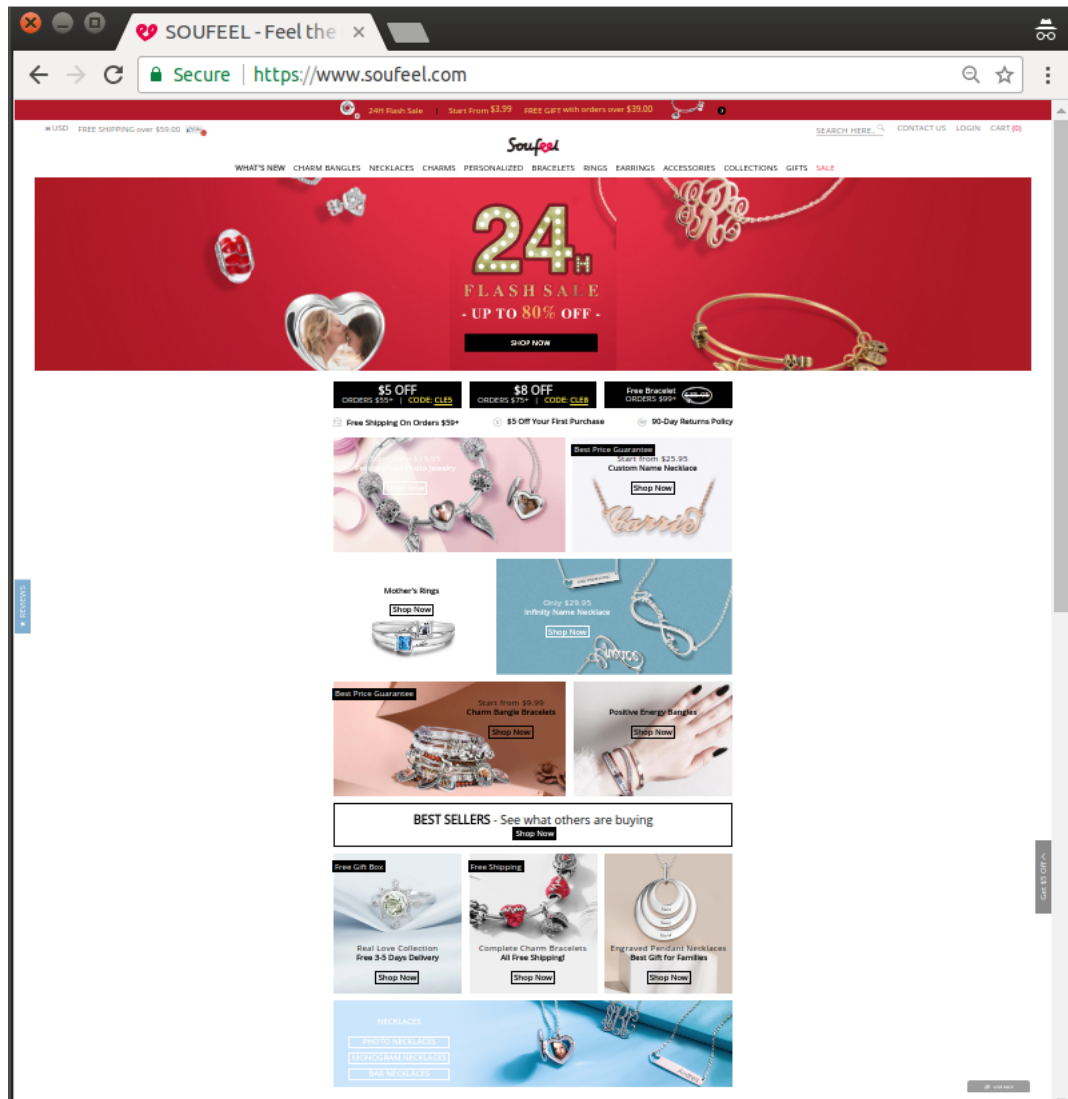
datasrc attribute

databgsrc attribute

Fig. 84. bbc.com lazy loading code. <http://web.archive.org/web/20180223143424/http://static.bbc.co.uk/news/1.230.02384/js/compiled/all.js>



(a) <http://www.soufeel.com/> as it exists archived. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>



(b) <http://www.soufeel.com/> as it exists on the live web

Fig. 85. <http://www.soufeel.com/> archived vs live web

As shown in Figure 85a, replay of the e-commerce site <http://www.soufeel.com/> is severely impacted (Figure 85b) by the lack of rewriting on the part of the Wayback Machine to URLs used for the pages image's, all of which are lazy loaded (Figure 86). What makes this page unique, beyond the fact that all of its images are lazy loaded, is that the page is using three different ways for lazy loading its images (Figures 88, 90, and 92).

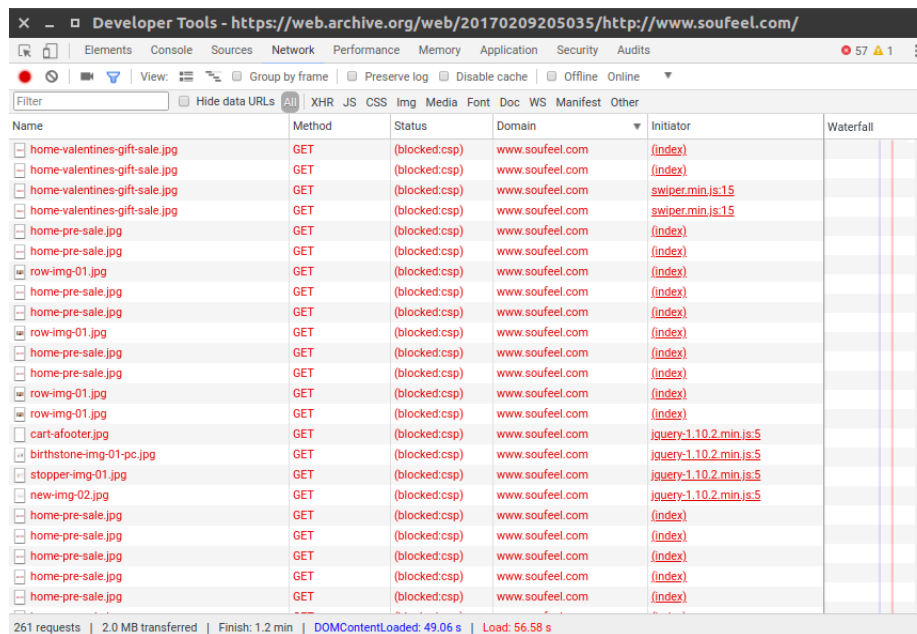


Fig. 86. Network tab of the developers console when replaying soufeel.com from the Wayback Machine displaying the blocked images. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>

```


<div class="swiper-slide lazy" data-lazystyle="width:100%; background:url('http://www.soufeel.co
↳ m/skin/frontend/smartwave/default/images/home/row-img-01.jpg') center center
↳ no-repeat;background-size:cover;"></div>
```

Fig. 87. Select HTML elements used by lazy loading way 1. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>

The first way (Figure 88) is used to lazy load both `img` and `div` tags matching those seen in Figure 87. When the `lazyRun` function is used, `img` tags with the `data-lazysrc` attribute have their `src` attribute set using the value of the `data-lazysrc` attribute and `div` tags with the `data-lazystyle` have their `style` attribute set to the value of the `data-lazystyle` attribute. Also, seen in Figure 88 is when a modal dialog is shown to the viewers of the page, the modal's `innerHTML` attribute is set using the query selector retrieved from the `data-body` attribute and an anchor tag has its `href` attribute set from its `data-href` attribute.

```

jQuery('#modal-confirm').on('show.bs.modal', function (e) {
  var target = jQuery(e.relatedTarget), modal = jQuery(this);
  if (target.data('body') != 'undefined') {
    var bodyId = target.data('body');
    modal.find('.modal-body').html(jQuery(bodyId).html());
  }
  if (target.data('href') != 'undefined') {
    var href = target.data('href');
    modal.find('.modal-footer .yes').attr('href', href);
  }
});
// code removed for presentation
function lazyRun(domElement) {
  var jdom = jQuery(domElement);
  // code removed for presentation
  if(jdom.data("lazysrc") != undefined) {
    jdom.attr('src',function(){
      var lazysrc = jdom.data("lazysrc");
      jQuery(this).removeAttr('data-lazysrc');
      jQuery(this).next('.swiper-lazy-preloader').remove();
      return lazysrc;
    })
  }
  if(jdom.data("lazystyle") != undefined) {
    var style = jdom.attr('style');
    style = style == undefined ? '' : style + ';';
    jdom.attr('style', function(index, oldvalue){
      var lazystyle = jdom.data("lazystyle");
      jQuery(this).next('.swiper-lazy-preloader').remove();
      jQuery(this).removeAttr('data-lazystyle');
      return style + lazystyle;
    })
  }
}

```

Sets element innerHTML from data-body

Sets element href attribute from data-href

Sets img src attribute from data-lazysrc

Sets element style attribute from data-lazystyle

Fig. 88. soufeel.com lazy loading way 1, code embedded in HTML and formatted for presentation. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>

The second way (Figure 90) is tied to when viewers of the page scroll down the page in order to load the `img` tags seen in Figure 89. When a scroll event is detected, the `img` tags with the *data-original* attribute have their *src* attribute set to the value of the tags *data-original* attribute. If the lazily loaded tag is not an `img` tag, then the tags *style* attribute has the property `background-image` set to the value of the tags *data-original* attribute.

```

<img class="lazyload"
data-original="http://static.soufeel.com/media/catalog/product/cache/0/thumbnail/280x280/9df78e
↳ ab33525d08d6e5fb8d27136e95/X/S/XS1074_1.jpg"/>
<img class="lazyload" id="product-collection-image-3440"
data-original="http://static.soufeel.com/media/catalog/product/cache/0/small_image/280x280/9df7
↳ 8eab33525d08d6e5fb8d27136e95/X/S/XS1401A.png"/>

```

Fig. 89. soufeel.com select HTML elements used by lazy loading way 2. Formatted for presentation. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>

```

a.fn.lazyload = function (f) {
  var h, i = this, j = {
    threshold: 0, failure_limit: 0, event: "scroll", effect: "show",
    container: b, data_attribute: "original", skip_invisible: !0,
    appear: null, load: null,
    placeholder: "data:image/png;base64,iVBORw0KGgoAAAANSUgAAAAEAAAABAQMAAAAL21bKAAAAA1BMV
    EX5+fkccGXJAAAACKlEQVR4XmNgAAAAAgAB3p6PvwAAAABJRU5ErkJggg=="
  },
  return (f && (d !== f.failurelimit && ((f.failure_limit = f.failurelimit),
  delete f.failurelimit),
  d !== f.effectspeed && ((f.effect_speed = f.effectspeed), delete
  f.effectspeed), a.extend(j, f)),
  (h = j.container === d || j.container === b ? e : a(j.container)), 0 ===
  j.event.indexOf("scroll") && h.bind(j.event, function () { return g();}),
  this.each(function () { var b = this, c = a(b);
  (b.loaded = !1), (c.attr("src") === d || c.attr("src") === !1) &&
  c.is("img") && c.attr("src", j.placeholder),
  c.one("appear", function () {
    if (!this.loaded) {
      if (j.appear) { var d = i.length; j.appear.call(b, d, j); }
      a("<img />").bind("load", function () {
        var d = c.attr("data-" + j.data_attribute);
        c.hide(), c.is("img") ? c.attr("src", d) :
        c.css("background-image", "url('" + d + "')"),
        c[j.effect](j.effect_speed), (b.loaded = !0);
        var e = a.grep(i, function (a) { return !a.loaded; });
        if (((i = a(e)), j.load)) { var f = i.length; j.load.call(b, f, j); }
        }).attr("src", c.attr("data-" + j.data_attribute));
      }}, 0 !== j.event.indexOf("scroll") && c.bind(j.event, function () {
        b.loaded || c.trigger("appear");
      }));
    }
  }
}

```

Data attribute for retrieving the URL

Blank placeholder

Set img src attribute to blank placeholder

if c is an img set src attribute to d

else set style attribute background-image to d

d = URL from data attribute

Fig. 90. soufeel.com lazy loading way 2 code. Formatted for presentation. <https://web.archive.org/web/20170209205035/http://static.soufeel.com/js/smartwave/jquery/plugins/lazyload/jquery.lazyload.min.js>

```

<a class="home-banner-img home_bg6 swiper-lazy"
  ↪ href="/web/20170209205035/http://www.soufeel.com/presale"
data-background="http://www.soufeel.com/skin/frontend/smartwave/default/custom/static/brand/act_
  ↪ ivity/presale/128/home-pre-sale.jpg"></a>
<a class="home-banner-img home_bg6 swiper-lazy"
  ↪ href="/web/20170209205035/http://www.soufeel.com/valentines-gift-sale"
data-background="http://www.soufeel.com/skin/frontend/smartwave/default/custom/static/brand/act_
  ↪ ivity/valentines-gift-sale/home-valentines-gift-sale.jpg"></a>

```

Fig. 91. soufeel.com select HTML elements used by lazy loading way 3. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>

The third and final way (Figure 92) is tied to the anchor tags (Figure 91) with the *data-background* attribute or elements with the *data-srcset* or *data-sizes* attributes contained in a slider. As the slider progresses, the images are loaded according the value of the anchor tag's data attribute, which in the case of <https://web.archive.org/web/20170209205035/http://www.soufeel.com/>, are anchor tags that have their *style* attribute's *background-image* property set to the value of the tags *data-background* attribute. The final set of well-known 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 (Figure 93).

```

S.lazy = {
  loadImageInSlide: function(e, t) {
    if ("undefined" != typeof e && ("undefined" == typeof t && (t = !0), 0 !==
    S.slides.length)) {
      var s = S.slides.eq(e), i =
      s.find("." + S.params.lazyLoadingClass + ":not(." + S.params.lazyStatusLoadedCla
      ss + "):not(." + S.params.lazyStatusLoadingClass + ")");
      !s.hasClass(S.params.lazyLoadingClass) ||
      s.hasClass(S.params.lazyStatusLoadedClass) ||
      s.hasClass(S.params.lazyStatusLoadingClass) || (i = i.add(s[0])), 0 !==
      i.length && Retrieve URL from data attribute
      i.each(function() { var e = a(this);
      e.addClass(S.params.lazyStatusLoadingClass);
      var i = e.attr("data-background"), r = e.attr("data-src"), n =
      e.attr("data-srcset"), o = e.attr("data-sizes"); Set style attribute
      S.loadImage(e[0], r || i, n, o, !1, function() { background-image
      if ((i ? (e.css("background-image", "url('" + i + "')"),
      e.removeAttr("data-background")) : (n
      Or set && (e.attr("srcset", n), e.removeAttr("data-srcset")), o &&
      img srcset (e.attr("sizes", o), Or set img
      attribute e.removeAttr("data-sizes")), r && (e.attr("src", r), src attribute
      e.removeAttr("data-src"))),

```

Fig. 92. soufeel.com lazy loading way 3 JavaScript code. <https://web.archive.org/web/20170209205035/http://static.soufeel.com/js/smartwave/jquery/plugins/swiper/js/swiper.min.js>

```

1  [Constructor]
2  interface XMLHttpRequest : XMLHttpRequestEventTarget {
3      void open(DOMString method, DOMString url);
4  };
5
6  [Unforgeable]
7  interface Location {
8      attribute DOMString href;
9      void assign(DOMString url);
10     void replace(DOMString url);
11 };
12
13 partial interface Window {
14     Promise<Response> fetch(RequestInfo input, optional Dictionary init);
15 };
16
17 typedef (Request or USVString) RequestInfo;
18
19 [Constructor(RequestInfo input, optional Dictionary requestInitDict)]
20 interface Request {
21     readonly attribute USVString url;
22 };
23
24 [Constructor(DOMString scriptURL)]
25 interface Worker : EventTarget {};
26
27 [Constructor(DOMString scriptURL, optional DOMString name)]
28 interface SharedWorker : EventTarget {};

```

Well-known URL identifiers

Well-known URL identifiers

URL

Type that is an interface with well-known URL identifier

Well-known URL identifier

Fig. 93. Well-known URL identifiers variations and patterns

As shown in Figure 93, the identifiers of non-element interface identifiers can be discovered by using the naming conventions of their attributes, and operation or constructor argument identifiers, namely *url* (lines 3, 9-10, and 21), *scriptURL* (lines 24 and 27), and *href* (line 8). The downside to using only the identifier names previously mentioned is we would miss the operation identifier `fetch` (Figure 93, line 13) 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 that is overcome when noticing the typing of `fetch`'s first argument `input` (line 14), which is `RequestInfo` (line 16), a typedef for the `Request` interface or a string. Because we have already 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` interfaces type (`Request`) to discover additional

identifiers. By matching on the type or typedef of an already identified interface, we discover the declared constructor of the `Request` interface and the `fetch` operation of the `Window` interface. More specifically, because the `url` attribute is `readonly` and the `Request` interface has a constructor whose arguments are an instance of itself or a string, we can conclude that the `url` attribute is set via one of the types in the typedef or by the `requestInitDict` (Figure 93, line 19). 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 finding discovering usages of the type for an already discovered interfaces (Figure 94).

```

1 interface Document : Node {
2   [PutForwards=href, Unforgeable] readonly attribute Location? location;
3 };
4
5 [PrimaryGlobal]
6 interface Window : EventTarget {
7   [PutForwards=href, Unforgeable] readonly attribute Location location;
8 };

```

Fig. 94. Type matching to discover additional interfaces which expose an attribute whose typing is an identified interface

The `Location` interface previously discovered by matching on the well-known non-element interface identifiers `href` and `url`, (Figure 93, lines 5-10) is also discoverable as the location attribute of the `Document` and `Window` interfaces when matching on the attribute's type (Figure 94, lines 2 and 7). Besides the `Unforgeable` extended attribute, (Figure 94, lines 2 and 7), the `location` attribute of both `Window` and `Document` has the `PutForwards=href` extended attribute defined for it and `href` is a well-known non-element interface identifier. But that presents a problem given the nature of that extended attribute, which is that it indicates that direct assignments to the attribute from the exposing interface, i.e. `window.location = "<URL>"`, is equivalent to directly assigning the target attribute (`href`), i.e. `window.location.href = "<URL>"` [18, 12].

Further more, compounding the problem is that both the `Document` and `Window` interfaces expose `Unforgeable` instances of the `Location` interface, and when an identifier has the `Unforgeable` extended attribute defined for it, we cannot override it [18]. Because the location interface can be used to navigate the browser and is an attribute of the primary global of the JavaScript execution environment object

Window, it will require special handling beyond identification. To illustrate this, consider the following example that uses the `Location` interface in conjunction with the `History` interface (Figure 95), which would have been discovered using the non-element well-known identifier `url`, to change the URL of displayed in the navigation bar of the browser when replaying a web page from the Internet Archive's Wayback Machine from a URI-M to a nonexistent URL (Figure 96 and Figure 97).

```

1 interface History {
2   void pushState(SerializedScriptValue data, DOMString? title, DOMString? url);
3   void replaceState(SerializedScriptValue data, DOMString? title, DOMString? url);
4 };

```

Fig. 95. History interface

Change The URL Of The Navigation Bar Without Navigating Away From The Page

Starting Location: `http://web.archive.org/web/20180224172547/http://www.cs.odu.edu/~jberlin/simpleHistory.html`
 Current Location: `http://web.archive.org/web/20180224172547/http://www.cs.odu.edu/~jberlin/simpleHistory.html`

`/ manipulating/browser/history/is/fun` Change It!

```

// the magic <3
document.getElementById('startingLoc').innerText = `Starting Location: ${window.location}`
document.getElementById('curLoc').innerText = `Current Location: ${window.location}`
const theInput = document.getElementById('changeHistory')
const doChange = document.getElementById('doChange')
doChange.onclick = () => {
  const value = theInput.value
  if (value === '') return
  if (Math.random() >= 0.5) {
    history.pushState(null, null, `${window.location.origin}/${value}`)
  } else {
    history.replaceState(null, null, `${window.location.origin}/${value}`)
  }
  document.getElementById('curLoc').innerText = `Current Location: ${window.location}`
  theInput.value = ''
}

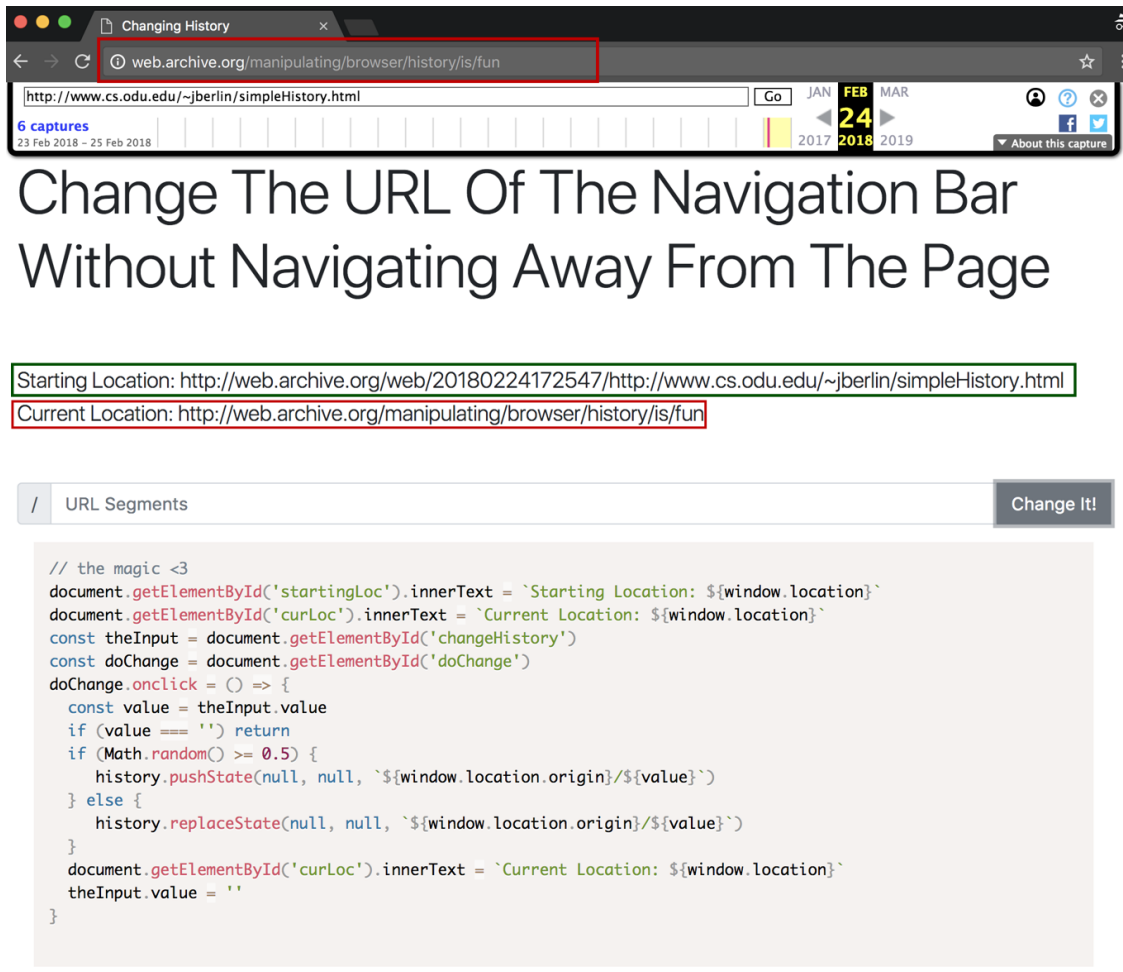
```

Fig. 96. Browser history manipulation using the `History` and `Location` interfaces before manipulation.

The example page seen in Figure 96 provides an input field for entering in URL path segments which are then appended to origin of the browser (accessed through `window.location.origin`) and used as the *url* argument of either the push or replace state operations of the history interface (Figure 95, lines 2-3) when the button labeled “Change It!” is pressed. The page also displays its starting location, set only once using the value of `window.location`, and the current location also set using `window.location` but changed every time the “Change It!” button is pressed. The results of entering “/manipulating/browser/history/is/fun” into the input field and pressing the “Change It!” button is seen in Figure 97. Because both the Location and History interfaces are not overridden, the resulting URL displayed is not as would be expected from the page’s live web behavior (Figure 98):

```
https://web.archive.org/manipulating/browser/history/is/fun
```

```
https://web.archive.org/web/20180224172547/http://www.cs.odu.edu/manipulating  
/browser/history/is/fun
```



Changing History

web.archive.org/manipulating/browser/history/is/fun

http://www.cs.odu.edu/~jberlin/simpleHistory.html

6 captures
23 Feb 2018 - 25 Feb 2018

Starting Location: <http://web.archive.org/web/20180224172547/http://www.cs.odu.edu/~jberlin/simpleHistory.html>

Current Location: <http://web.archive.org/manipulating/browser/history/is/fun>

/ URL Segments Change It!

```
// the magic <3
document.getElementById('startingLoc').innerText = `Starting Location: ${window.location}`
document.getElementById('curLoc').innerText = `Current Location: ${window.location}`
const theInput = document.getElementById('changeHistory')
const doChange = document.getElementById('doChange')
doChange.onclick = () => {
  const value = theInput.value
  if (value === '') return
  if (Math.random() >= 0.5) {
    history.pushState(null, null, `${window.location.origin}/${value}`)
  } else {
    history.replaceState(null, null, `${window.location.origin}/${value}`)
  }
  document.getElementById('curLoc').innerText = `Current Location: ${window.location}`
  theInput.value = ''
}
```

Fig. 97. Browser history manipulation using the History and Location interfaces after manipulation, <http://web.archive.org/web/20180224172547/http://www.cs.odu.edu/~jberlin/simpleHistory.html>

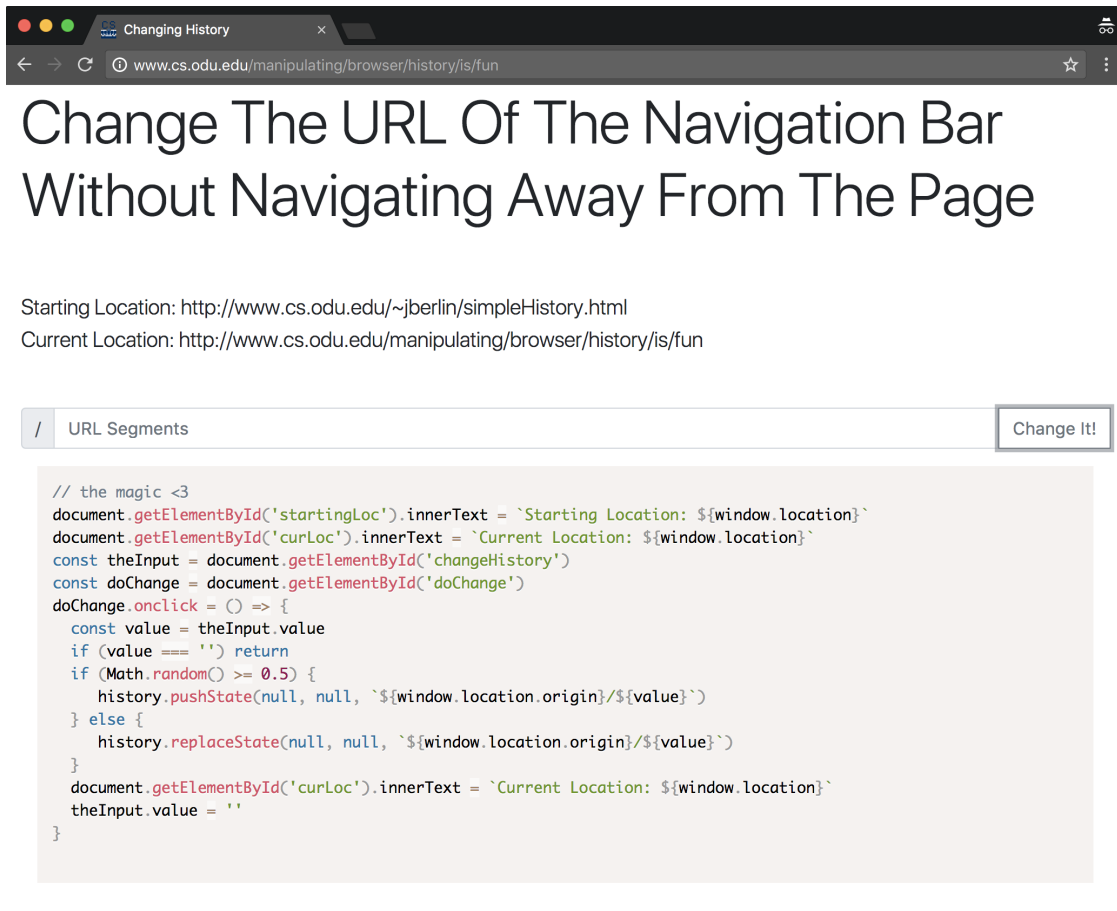


Fig. 98. Browser history manipulation using the History and Location interfaces live web, <http://www.cs.odu.edu/~jberlin/simpleHistory.html>

Finally, recall the replay issues discussed in the introduction of this thesis for mementos of CNN [11] and user pages found at mendeley.com [15]. The CNN webpage could not replay because the archived JavaScript wished to set the domain of the page to one that is different than the archives, and Mendeley user pages needed cookies, both of which are properties of the `Document` interface and should be added to well-known identifiers. This leaves us with a baseline of ten identifiers, seven for discovering interfaces that are `HTMLElements` and three identifiers for the discovery of non-`HTMLElement` interfaces (Table 10). The remaining well-known interface and member identifiers found in Table 11 are considered special cases given the specificity of their use cases.

Table 10
Baseline Interface Discovery Identifiers

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

Table 11
Special Well-Known Interface and Member Identifiers

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

5.3.2 AUTOMATIC WEB IDL INTERFACE IDENTIFICATION

Automatically identifying relevant Web IDL interfaces can be expressed in two phases: fragment extraction (Algorithm 1) and identification (Algorithm 2). In the first phase, fragment extraction, each fragment's members from the set of Web IDL fragments to be considered are extracted and the following steps are performed.

Algorithm 1 Web IDL Fragment extraction

```

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

```

For each member, we accumulate a mapping of interface identifiers to interfaces, which interfaces an interface implements, ensure 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 they are part of. Finally, this returns the interfaces and typedefs extracted to the next phase identification.

Algorithm 2 Identify interfaces

```

1: function IDENTIFYINTERFACES(idlFragments)
2:   interfaces, typeDefs, isTypeDefd  $\leftarrow$  GetIdlData(idlFragments)
3:   foundInterfaces  $\leftarrow$  Map
4:   for each interface  $\in$  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(inteface,specialCheck,foundInterfaces)
12:    for each foundId  $\in$  foundInterfaces.keys do
13:      if isTypeDefd.hasKey(foundId) then
14:        FindTypedefArugments(isTypeDefd[foundId],interfaces,foundInterfaces)
15:    for each foundId  $\in$  foundInterfaces.keys do
16:      if nonElementInterface(foundInterfaces[foundId]) then
17:        CheckFoundAttsRefFoundType(foundId,interfaces,foundInterfaces)
18:    return foundInterfaces,typeDefs,isTypeDefd

```

The identification phase (Algorithm 2) is multi-part algorithm performing the steps for identifier discovery as described in the previous section, Subsection 5.3.1. For each interface that has an interface object, we check to see if it has members matching the well-known HTML identifiers if it is an HTMLElement, otherwise the check is performed using the non-element identifiers (Algorithm 3). A further check is made to determine if the interface is a special case and if so, its members are checked using the associated attribute or operation identifiers found in Table 11 (Algorithm 3).

Algorithm 3 Check Interface Based On Identifier Names

```

1: function CHECKMEMBERIDENTIFIERS(interface,toFind,found)
2:   attributes  $\leftarrow$  getAttributesNamed(interface,toFind)
3:   operations  $\leftarrow$  getOperationsWithArgsNamed(interface,toFind)
4:   constructors  $\leftarrow$  getConstructorsWithArgsNamed(interface,toFind)
5:   if anyNotEmpty(attributes,operations,constructors) then
6:     identified  $\leftarrow$  newIdentified(interface,attributes,operations,constructors)
7:     found[identified.identifier]  $\leftarrow$  identified
8: function SPECIALCHECK(interface,specialCheck,found)
9:   attributes  $\leftarrow$  getAttributesNamed(interface,specialCheck.attributes)
10:  operations  $\leftarrow$  getOperationsWithArgsNamed(interface,specialCheck.operations)
11:  if anyNotEmpty(attributes,operations,constructors) then
12:    if found.hasKey(interface.identifier) then
13:      foundIface  $\leftarrow$  found[interface.identifier]
14:      foundIface.attributes  $\leftarrow$  foundIface.constructors  $\cup$  attributes
15:      foundIface.operations  $\leftarrow$  foundIface.operations  $\cup$  operations
16:    else
17:      foundIface  $\leftarrow$  newIdentified(interface,attributes,operations)
18:      found[identified.identifier]  $\leftarrow$  identified

```

Once every interface having an interface object has been checked, we determine which of the identified interfaces are used in a typedef and check the arguments of operations and constructor of each identified interface to determine if their typing is the typedef. Then we 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). Finally, this returns the identified interfaces and the typedef information for use in code generation.

Algorithm 4 Find Interfaces with members typed

```

1: function FINDTYPEDEFARUGMENTS(typeDefs,interfaces,found)
2:   for each typedef  $\in$  typeDefs do
3:     for each interface  $\in$  interfaces do
4:       operations  $\leftarrow$  getOperationsWithArgsTyped(interface,typedef)
5:       constructors  $\leftarrow$  getConstructorsWithArgsTyped(interface,typedef)
6:       if operations  $\neq \emptyset \vee$  constructors  $\neq \emptyset$  then
7:         if found.hasKey(interface.identifier) then
8:           foundIface  $\leftarrow$  found[interface.identifier]
9:           foundIface.operations  $\leftarrow$  foundIface.operations  $\cup$  operations
10:          foundIface.constructors  $\leftarrow$  foundIface.constructors  $\cup$  constructors
11:         else
12:           foundIface  $\leftarrow$  newIdentified(interface, $\emptyset$ ,operations,constructors)
13:           found[foundIface.identifier]  $\leftarrow$  foundIface
14: function CHECKFOUNDATTSREFFOUNDTYPE(foundId,interfaces,found)
15:   for each fld  $\in$  foundInterfaces.keys do
16:     atts  $\leftarrow$  getAttributeOfType(interfaces[fld],foundId)
17:     if notEmpty(atts) then
18:       foundInterfaces[fld].exposesFound[foundId]  $\leftarrow$  atts
19:       exposed  $\leftarrow$  foundInterfaces[foundId]
20:       exposed.exposedOnFound  $\leftarrow$  exposed.exposedOnFound  $\cup$  fld

```

The identification algorithms described was run on an input set of Web IDL fragments retrieved from the source code repository of the Chromium browser¹ using the W3C provided Node.js parser². Unsurprisingly, the HTML element interfaces which we are interested in were discovered as was the named constructor `Audio` associated with the `HTMLAudioElement` (Table 12). We did not re-list the special case interfaces as they have already been shown to exist (Figures 72 and 76) and listed previously (Table 11).

¹<https://chromium.googlesource.com/chromium/blink/+master/Source>

²<https://github.com/w3c/webidl2.js>

Table 12
Identified HTML Interfaces

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

The remaining interfaces and attribute or operation identifiers were discovered using the well-known non-element identifiers from Table 10 and the type matching heuristic (Table 13). The identification algorithm was successfully able to identify the interfaces for using the HTTP protocol namely `fetch`, `Request`, `Response`, `XMLHttpRequest` and `EventSource` [12], the WebSocket protocol [71], as well as the location attribute of both `Window` and `Document`.

Table 13
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
		sendBeacon, registerProtocolHandler
Navigator	Operations	isProtocolHandlerRegistered
		unregisterProtocolHandler
	Attributes	url
Request	Constructors	Constructor(input, requestInitDict)
	Attributes	url
Response	Operations	redirect
	Operations	register
ServiceWorkerContainer	Operations	fetch
ServiceWorkerGlobalScope	Operations	fetch
SharedWorker	Constructors	Constructor(scriptURL, name)
	Attributes	href
URL	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

5.3.3 REWRITER MODIFICATIONS

Generation of the client-side rewriting library, although informed by the identified interfaces, relies on the Web IDL to JavaScript mapping discussed in Section 5.2 as the basis for generating the overrides applied to those interfaces. As discussed in Section 5.2, we know that 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 [31, 18]. 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. Explicit creation of objects is done via function objects, which act as the constructor for the object [31]. That is to say that every object in JavaScript has a prototype but may not have a function object as seen in Figures 99 and 100.

```

1 interface HTMLIFrameElement : HTMLElement {← Object
2   attribute DOMString srcdoc; ← Attribute on prototype
3 };
4
5 [Constructor] ← Function object
6 interface XMLHttpRequest : XMLHttpRequestEventTarget {
7   void open(DOMString method, DOMString url);
8 };
9
10 ← Operation on prototype

```

Fig. 99. Object vs Function Object Web IDL

The differentiation between whether an interface is a function object or an object is that interface which have the `Constructor` extended attribute defined for it are function objects and interfaces lacking the `Constructor` extended attribute are not as seen in Figure 99. Because the `HTMLIFrameElement` interface lacks the definition of the `Constructor` extended attribute (line 1), you cannot create a new instance of the interface directly (Figure 100, line 2), whereas because the `XMLHttpRequest` interface is defined with the `Constructor` extended attribute (line 5) you can (Figure 100, line 7). But both new instances have a prototype object (Figure 100, lines 4 and 9) that allows them to behave as defined by their respective Web IDL definitions (Figure 99).

```

1 // object that is not a function object but is creatable
2 let iframe = document.createElement('iframe')
3 // mutate property on prototype for this instance
4 iframe.srcdoc = '<a href="http://abc.xzy">Hi</a>'
5
6 // object that is a function object creatable using constructor
7 let httpRequest = new XMLHttpRequest()
8 // call function property on prototype for this instance
9 httpRequest.open('GET', 'https://google.com')
```

Fig. 100. Object vs. Function Object JavaScript

It must be noted that all interfaces representing HTML elements are not function objects and cannot be created directly using the constructor pattern of function objects, because they are associated with the HTML markup of the document [30]. Hence, there is the restriction of only being creatable using the `createElement` operation of the `Document` interface (Figure 100, line 2), except for `HTMLElement` interfaces that have the `NamedConstructor` extended attributed defined (Figures 101 and 102).

```

1 [Exposed=Window,NamedConstructor=Audio(DOMString src)]
2 interface HTMLAudioElement : HTMLMediaElement {
3     attribute DOMString src;
4 };
```

Fig. 101. HTMLAudioElement named constructor Web IDL

The named constructor of the `HTMLAudioElement` interface `Audio` (Figure 101, line 1), although it may appear to be a separate function object that inherits from `HTMLAudioElement`, it is not a separate function object. The HTML specification considers the audio element to represent a sound or audio stream that is not required to exist inside the HTML markup of the document for instances of the interface to be used to play audio [12]. Because the HTML specification does not restrict the usage of the audio element to only functioning as an HTML element, you can create a new HTML audio element using the `Audio` named constructor (Figure 102, line 2) and begin playing audio without adding the new element to the document (Figure 102, line 3).

```

1 // named constructor to create new HTMLAudioElement interface object
2 let audio = new Audio('http://music.com/burial/kindred.mp3')
3 audio.play()

```

Fig. 102. HTMLAudioElement named constructor JavaScript

Since we know that every identified interface will have a **prototype object** but not necessarily a function object, and the **prototype object** provides a permanent record of its properties, 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, as the name implies, patches the **prototype object** of an identified interface that does not expose a constructor by redefining the named properties of the interface's attributes and operations in order to intercept un-rewritten URLs (Figure 103). Because the **prototype object** provides a permanent record for all properties an object has, redefinition of attributes only replaces the original named property's 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 archived controlled version (Figure 104).

```

1 interface Element : Node {
2   attribute DOMString innerHTML; } Redefine properties getter and setter
3   attribute DOMString outerHTML; } functions on prototype object
4
5   DOMString? getAttribute(DOMString name); } Redefine function directly
6   void setAttribute(DOMString name, DOMString value); } on prototype object
7 };

```

Fig. 103. Interface attribute and operation patch overrides

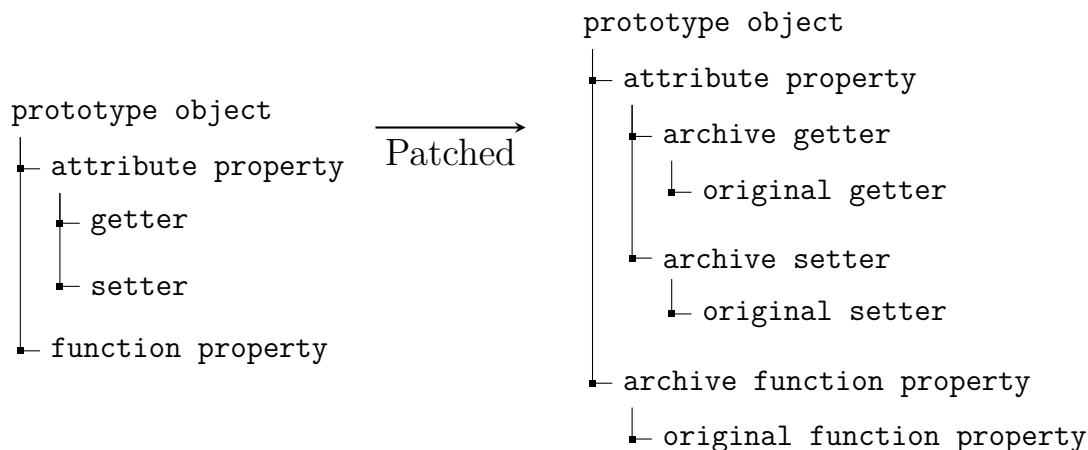


Fig. 104. Prototype object patch modification

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 (Figure 105). Because the `Window` interface is the global execution object, it represents the current browsing context, and with the in-place security constraints of the browser, we cannot directly modify the existing `Window` object or its prototype. Rather, the existing instance of the `Window` interface (`window`) is a `WindowProxy` object [12, 72] that proxies the exposed attributes and operations of the `Window` interface. Both are internally managed by the browser. In order to ensure that the attributes and operations exposed by the `Window` interface cannot be used to introduce un-rewritten URLs, we can only replace their definitions with archived-controlled versions on the `WindowProxy` object (Figure 106). Unlike the `Window` interface, the additional interface which has an existing instance can have the overrides targeting their exposed attributes and operations made to both the existing instance and prototype object.

```

1 partial interface Window {
2     Promise<Response> fetch(RequestInfo input, optional Dictionary init);
3 };

```

← Replace definitions on existing instance

Fig. 105. Interface existing instance replace modification

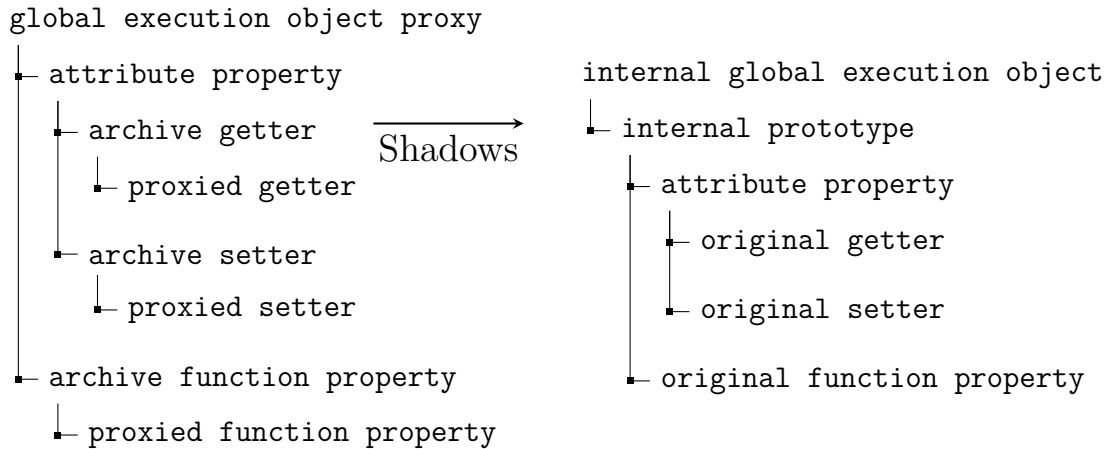


Fig. 106. Global execution object replace modification

The replace plus patch override, as the name implies, 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` (Figure 107, lines 2-3). We wish to ensure that both the existing instance and its prototype object share the same overrides that were made to the existing instance (Figure 108). By replacing the existing instance's copy of the operations and patching the prototype object for the interface, we ensure that no reference to an un-patched version of the named property exposed by the interface can be retrieved. The next override, foreign substitution, unlike the other overrides, is the only override to introduce a new (foreign) representation of the interfaces it is targeting.

```

1 interface Document : Node {
2     attribute DOMString domain;
3     void write(DOMString... text);
4     void writeln(DOMString... text);
5 };

```

} Replace and patch
defintions on
existing instance
and prototype object

Fig. 107. Existing instance replace plus patch

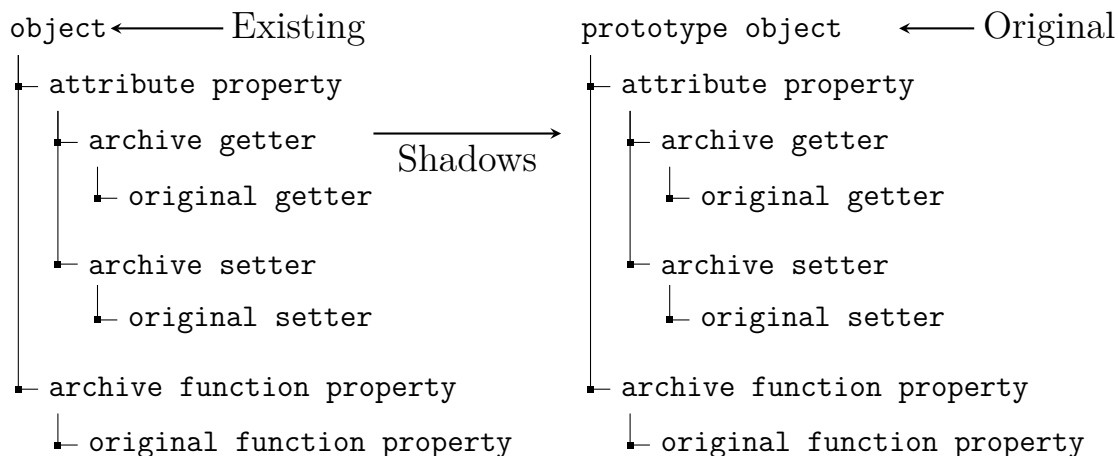


Fig. 108. Existing instance replace plus patch modification

The foreign substitution override exists primarily due to the capabilities of the unforgeable (unmodifiable or overridable) `Location` interface which is also an `Unforgeable` attribute of the primary global object `Window` and the `Document` interface (Figure 109), discussed previously in Sections 5.3.1 and 5.2. Any assignment to the existing instance of the `Location` interface itself or to the interfaces `href` attribute will navigate the browser away from the current page as well as usage of its two operations, `assign` and `replace`. Because of the capabilities of the unforgeable `Location` interface, archives began rewriting server-side the text string “location” found in the archived JavaScript along-side any URLs it could identify to reference `WB_wombat_self_location`, an archive implementation of the `Location` interface (Figure 47), discussed in the temporal jailing section of Chapter 4. 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, it was also rewriting instances of the text string “location” that were not actual nor could ever be instances of the `Location` interface (Figure 110).


```

[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. 109. Foreign substitution unforgeable Location interface

```

1 // archive it version // live web version
2 window.__PRELOADED_STATE__ = { window.__PRELOADED_STATE__ = {
3   "WB_wombat_self_location": { "location": {
4     "id": "fcc2fd44-d82e-45ca-8855-35ee6b8bfbe9", "id": "fcc2fd44-d82e-45ca-8855-35ee6b8bfbe9",
5     "latitude": 63.44, "latitude": 63.44,
6     "longitude": 10.4, "longitude": 10.4,
7     "name": "Trondheim, Norway", "name": "Trondheim, Norway",
8     "city": "Trondheim", "city": "Trondheim",
9     "state": "Sør-Trøndelag", "state": "Sør-Trøndelag",
10    "country": "Norway" "country": "Norway"
11  },
12  };
13
14  o.Auth.authCodeFlow({ o.Auth.authCodeFlow({
15    authenticateOnStart: !1, authenticateOnStart: !1,
16    apiAuthenticateUrl: function() { apiAuthenticateUrl: function() {
17      var t = "/sign-in/?routeTo=" + encodeURIComponent(WB_wombat_self_location); var t = "/sign-in/?routeTo=" +
18      encodeURIComponent(WB_wombat_self_location); encodeURIComponent(location);
19      return WB_wombat_self_location = t; return location = t
20    },
21    refreshAccessTokenUrl: refreshAccessTokenUrl:
22    ↪ "/profiles/refreshToken/" ↪ "/profiles/refreshToken/"
23  });
24
25  function s(t) { function s(t) {
26    var e = t.headers.WB_wombat_self_location; var e = t.headers.location;
27    if (e && this.settings.followLocation && if (e && this.settings.followLocation &&
28    201 === t.status) { 201 === t.status) {
29      var n = {method: "GET",url: e,responseType: var n =
30      ↪ "json"}; {method: "GET",url: e,responseType:
31      ↪ "json"};
32    return this.send(n); return this.send(n);
33  } }
34  return t.headers.link && "string" == typeof return t.headers.link && "string" == typeof
35  ↪ t.headers.link ↪ t.headers.link
36  && (t.headers.link = 1(t.headers.link)), t; && (t.headers.link = 1(t.headers.link)), t;
37  } }

```

Actual location

Not actual location

Not actual location

Fig. 110. Archive-It rewriting the text string “location” in the archived JavaScript of mendeley.com user pages. Live web version on the right.

The JavaScript code excerpts shown in Figure 110 come from the same Archive-It archived user page of `mendeley.com` that suffers from the infinite redirection issue on replay discussed in the introduction of this thesis (Chapter 1). As shown in Figure 110, Archive-It is only changing the text string “location” to “WB_wombat_self_location” correctly twice out of the four rewrites shown. The first incorrect rewrite happens for an object 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 but also in non-JavaScript content bundled with the JavaScript (Figure 111).

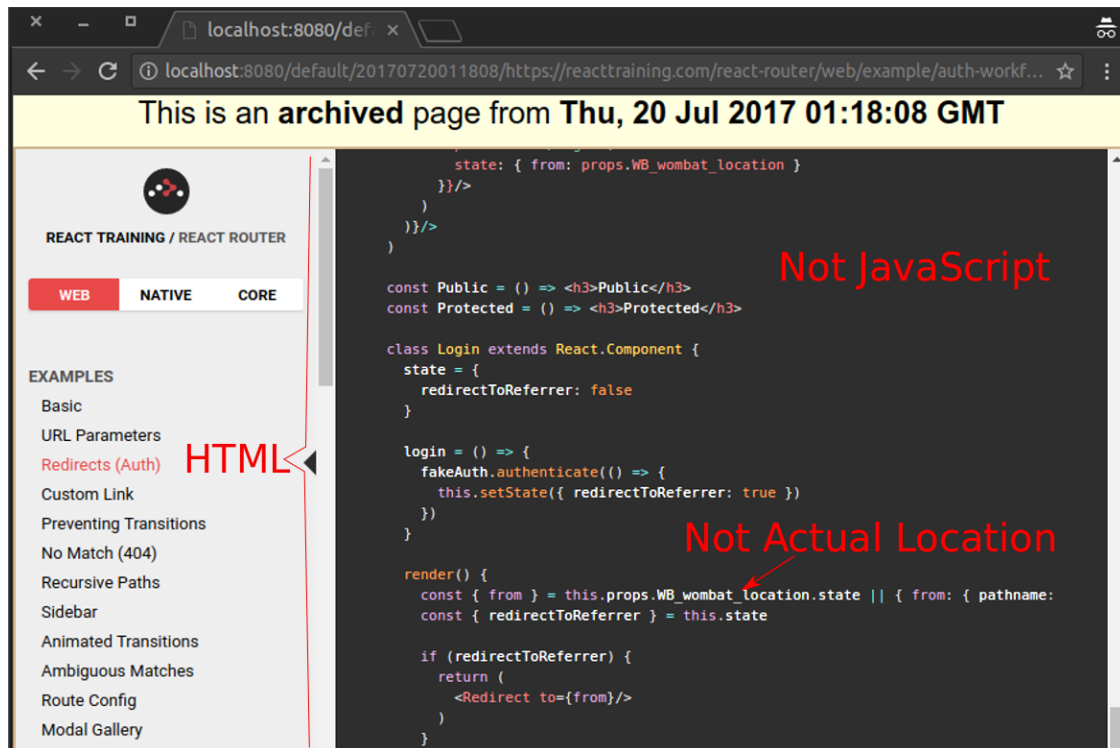


Fig. 111. Pywb version 0.33 rewriting the text string “location” found in non-JavaScript page markup for the documentation of React Router. <https://reacttraining.com/react-router/web/example/auth-workflow>

```

1  webpackJsonp([15], {
2    528: function(n, s) {
3      n.exports = '....bundled html removed for presentation....<span
•      class="token keyword">const</span> <span class="token
•      punctuation">{</span> <span class="token keyword">from</span> <span
•      class="token punctuation">}</span> <span class="token
•      operator">=</span> <span class="token keyword">this</span><span
•      class="token punctuation">.</span>props<span class="token
•      class="token punctuation">.</span>location<span class="token
•      punctuation">.</span>state <span class="token operator">||</span> <span
•      class="token punctuation">{</span> <span class="token
•      keyword">from</span><span class="token punctuation">:</span> <span
•      class="token punctuation">...bundled html removed for presentation...'
4    }
5  });

```

Fig. 112. Text string “location” in HTML bundled with the JavaScript for the documentation of React Router. <https://reacttraining.com/react-router/>

The page seen in Figure 111 is from the documentation for the JavaScript library React-Router³, which bundles the example code for its documentation as HTML strings alongside the page’s JavaScript. This allows for the pre-rendered documentation to be swapped out quickly rather than having to incur the cost of both parsing and rendering the JavaScript code. But because the example code was bundled with the page’s JavaScript, the text string “location” was incorrectly rewritten server-side due to the MIME type being *application/javascript* not *text/html*. A further example of the potentially destructive nature of server-side rewriting targeting non-URL text strings in archived JavaScript can be seen in Figure 113a. In addition to the rewriting of the text string “location”, the text string “top” was also being rewritten in order to ensure the archived JavaScript could not detect it was replayed from within an iframe, discussed in the Temporal Jailing section of Chapter 4. Comparing the documentation snippet from the archived page (Figure 113a) to its live web counterpart (Figure 113b), it is clear to see that those rewrites were made incorrectly.

³<https://github.com/ReactTraining/react-router>

Scroll to `WB_wombat_top`

Most of the time all you need is to “scroll to the `WB_wombat_top`” because you have a long content page, that when navigated to, stays scrolled down. This is straightforward to handle with a `<ScrollToTop>` component that will scroll the window up on every navigation, make sure to wrap it in `withRouter` to give it access to the router’s props:

```
class ScrollToTop extends Component {
  componentDidUpdate(prevProps) {
    if (this.props.WB_wombat_location !== prevProps.WB_wombat_location) {
      window.scrollTo(0, 0)
    }
  }

  render() {
    return this.props.children
  }
}

export default withRouter(ScrollToTop)
```

Then render it at the `WB_wombat_top` of your app, but below Router

- (a) Pywb Version 0.33 rewriting the text strings “location” and “top”

Scroll to top

Most of the time all you need is to “scroll to the top” because you have a long content page, that when navigated to, stays scrolled down. This is straightforward to handle with a `<ScrollToTop>` component that will scroll the window up on every navigation, make sure to wrap it in `withRouter` to give it access to the router’s props:

```
class ScrollToTop extends Component {
  componentDidUpdate(prevProps) {
    if (this.props.location !== prevProps.location) {
      window.scrollTo(0, 0)
    }
  }
}

render() {
  return this.props.children
}
}

export default withRouter(ScrollToTop)
```

Then render it at the top of your app, but below Router

(b) Original React Router documentation

Fig. 113. Incorrect rewriting of the text string “location” and “top” in React Router’s documentation. The incorrect rewrites are highlighted in red (Figure 113a) and the original strings in the documentation are highlighted in green (Figure 113b)

```
def_prop(win.Object.prototype, "WB_wombat_location", setter, getter);
def_prop($wbwindow.Object.prototype, "WB_wombat_top", setter, getter);
```

Fig. 114. Webrecorder and Pywb global patch override for its rewriting of the text strings “location” and “top”

```

// implicitly added to global window interface object
var WB_wombat_self_location;
var WB_wombat_top_location;
// added directly to document interface object
document.WB_wombat_self_location = WB_wombat_self_location;

```

Fig. 115. Archive-It’s mitigation for its rewriting of the text string “location” and “top”

In order to compensate for the server-side rewriting of the text strings “location” and “top” in archived JavaScript, Webrecorder and Pywb applied a patch override to the prototype inherited by every JavaScript object (Figure 114). Archive-It, on the other hand, is only adding the necessary objects as properties of the document and window interfaces (Figure 115). This implies that if the JavaScript code for handling HTTP request redirection, seen in Figure 110, were to be executed on replay, the code would fail because the location property was rewritten to `WB_wombat_self_location`, a non-existent property on an object representing HTTP headers. Obviously this is not correct rewriting on the part of Archive-It, Webrecorder, and Pywb. Even though the latter two correct this mistake by performing a global patch modification (Figure 114), all three would still perform the incorrect rewriting shown in Figures 111 and 113a.

The final override type, `extend`, creates a new subtype of non-element interfaces that have a constructor (Figure 99, lines 5-8) or an `HTMLElement` which has a named constructor (Figure 101) and replaces the reference to the interface on the primary global object with the archived controlled subtype. This override is different from foreign substitution as the new interface inherits all the properties of the extended interface and is a subtype of the extended interface, not a new foreign type (Figure 116).

```

1  class ArchiveControlledXMLHttpRequest extends XMLHttpRequest {
2    open (method, url, async, user, password) {
3      // example rewrite logic to demonstrate archive control
4      let maybeRewrittenURL = rewriter.rewrite(url)
5      // uses original XMLHttpRequest.open operation (inherited)
6      super.open(method, maybeRewrittenURL, async, user, password)
7    }
8  }
9  window.XMLHttpRequest = ArchiveControlledXMLHttpRequest

```

Fig. 116. Example extend override

5.3.4 REDUCING THE AMOUNT OF SERVER-SIDE REWRITING REQUIRED FOR THE FOREIGN SUBSTITUTION MODIFICATION

Due to the potentially dangerous server-side rewriting done in order to facilitate the foreign substitution modification, 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 [31].

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 three modifications via interceptors. This is especially useful for the creating a more thorough override for both the `Window` and `Document` interfaces (Figure 117), both of which had properties that were incorrectly rewritten (Figures 110, 111, and 113a). 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 (Figure 117, line 2) [31, 18, 12].

```

1 // window proxy
2 new window.Proxy({}, {
3   get (target, prop) { /* intercept attribute getter calls */,
4     set (target, prop, value) { /* intercept attribute setter calls */,
5     has (target, prop) { /* intercept attribute lookup */,
6     ownKeys (target) { /* intercept own property lookup */,
7     getOwnPropertyDescriptor (target, key) { /* intercept descriptor lookup */,
8     getPrototypeOf (target) { /* intercept prototype retrieval */,
9     setPrototypeOf (target, newProto) { /* intercept changes of prototype */,
10    isExtensible (target) { /* intercept is object extendable lookup */,
11    preventExtensions (target) { /* intercept prevent extension calls */,
12    deleteProperty (target, prop) { /* intercept is property deletion */,
13    defineProperty (target, prop, desc) { /* intercept new property definition */,
14  })
15
16 // document proxy
17 new window.Proxy(window.document, {
18   get (target, prop) { /* intercept attribute getter calls */,
19   set (target, prop, value) { /* intercept attribute setter calls */,
20 })

```

Fig. 117. Archive Window and Document interface proxies

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 117 lines 3-13. The existing instance for the `Document` interface on the other hand has no such restrictions and 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 117). Thus reducing the minimal server-side rewriting requirement for JavaScript to URL rewriting and the setup shown in Figure 118. 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.


```

1  var __archive$assign$function__ = function(name) {/*return archive override*/};
2  {
3    // archive overrides shadow these interfaces
4    let window = __archive$assign$function__("window");
5    let self = __archive$assign$function__("self");
6    let document = __archive$assign$function__("document");
7    let location = __archive$assign$function__("location");
8    let top = __archive$assign$function__("top");
9    let parent = __archive$assign$function__("parent");
10   let frames = __archive$assign$function__("frames");
11   let opener = __archive$assign$function__("opener");
12   /* archived JavaScript */
13  }

```

Fig. 118. Archive JavaScript proxy setup anonymous block scope

Each re-declaration (foreign substitution) is bound to the scope of the anonymous block wrapping the archived JavaScript, allowing for the re-declarations to shadow the originals from within the anonymous block scope only. This allows the archived JavaScript to be executed as if it were not wrapped inside the anonymous block unless it is operating in strict mode denoted by a string sentinel `"use strict"`, found as the first statement of the code. The `"use strict"` statement must be wrapped along with the replayed JavaScript in order for replay to not be impacted. When JavaScript code is not executed in strict mode, function definitions are lifted to the top most scope by the JavaScript runtime, causing them to become defined on the global object implicitly.

This implicit operation performed by the JavaScript runtime is required in order to use the anonymous scope to override the `Window` and `Document` interfaces while allowing all other JavaScript semantics to operate as expected in strict mode [31, 18, 12]. If an archive were to use a scope provided by a self-executing function (Figure 119), each re-declaration (foreign substitution) and the archived JavaScript code would be bound to scope of the function only. This would require further modifications on the part of the archive in order to ensure replay was not impacted which is undesirable [31].

```

1  var __archive$assign$function__ = function(name) {/*return archive override*/}
2  (function () {
3      // archive overrides of known interfaces with existing instances
4      let window = __archive$assign$function__("window");
5      let self = __archive$assign$function__("self");
6      let document = __archive$assign$function__("document");
7      let location = __archive$assign$function__("location");
8      let top = __archive$assign$function__("top");
9      let parent = __archive$assign$function__("parent");
10     let frames = __archive$assign$function__("frames");
11     let opener = __archive$assign$function__("opener");
12     /* archived JavaScript */
13 })();

```

Fig. 119. Archive JavaScript proxy setup function scope

As shown by the compatibility tables for the `let` declarator (Figure 120a), and JavaScript proxy object (Figure 120b) the minimal browser support for this method of performing the foreign substitution modification is Firefox v44, Chrome v49, Safari v10, Opera v39, and Microsoft Edge v12. Note that this method for performing the foreign substitution modification and its initial implementation were created as a part of the research for this thesis and contributed back to Pywb on April 28, 2017⁴. Both Webrecorder and Pywb have since fully adopted this method and has been using it in production since August 21, 2017⁵. We include it in this thesis so that it may be used by other archives to improve their replay fidelity and allow any archived JavaScript similar to Figures 121a and 121b to be replayed without modification.

	Desktop						Mobile						TV	
	Chrome	Edge	Firefox	Opera	Safari	Internet Explorer	Android	Chrome	Edge	Firefox	Opera	Safari	Internet Explorer	TV
Basic support	41	12	44 *	11	17	10	41	41	12	44 *	17	10	?	Yes

Full support Compatibility unknown
 * See implementation notes.

(a) Let declarator compatibility table. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let>

⁴<https://github.com/webrecorder/pywb/pull/215>

⁵<https://github.com/webrecorder/webrecorder/commit/12e2b507b88c4f0c00f29589436f7385dc512f9a>

	Desktop						Mobile						TV	
	Ch	E	FF	Ge	O	S	And	Ch	E	FF	O	S		TV
Basic support	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
revocable	Yes	Yes	34	No	Yes	10	Yes	Yes	Yes	34	Yes	10	?	Yes
handler.apply	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.construct	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.defineProperty	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.deleteProperty	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.enumerate	No	No	37 — 47	No	No	No	No	No	No	37 — 47	No	No	?	No
handler.get	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.getOwnPropertyDescriptor	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.getPrototypeOf	No	No	49	No	No	No	No	No	No	49	No	No	?	No
handler.has	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.isExtensible	?	?	31	No	?	?	?	?	?	31	?	?	?	?
handler.ownKeys	49	12	18 *	No	36	10	49	49	Yes	18 *	36	10	?	Yes
handler.preventExtensions	49	12	22	No	36	10	49	49	Yes	22	36	10	?	Yes
handler.set	49	12	18	No	36	10	49	49	Yes	18	36	10	?	Yes
handler.setPrototypeOf	?	?	49	No	?	?	?	?	?	49	?	?	?	?

- Full support - No support
- Compatibility unknown Non-standard. Expect poor cross-browser support.
 Deprecated. Not for use in new websites. * See implementation notes.

(b) JavaScript proxy compatibility table. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy

Fig. 120. Browser compatibility tables for the let declarator and JavaScript Proxy object

```

var _0xc76c = [
  "\x6F\x72\x69\x67\x69\x6E", "\x6C\x6F\x63\x61\x74\x69\x6F\x6E",
  "\x70\x75\x62\x6C\x69\x63\x6F\x2E\x70\x74",
  "\x69\x6E\x64\x65\x78\x4F\x66", "\x68\x72\x65\x66",
  "\x68\x74\x74\x70\x3A\x2F\x2F\x77\x77\x77\x2E\x70\x75\x62\x6C\x69\x63\x6F\x74"
  ↪ 2E\x70\x74"
];

```

```

if (window[_0xc76c[1]][_0xc76c[0]] &&
    window[_0xc76c[1]][_0xc76c[0]][_0xc76c[3]](_0xc76c[2]) < 0) {
    window[_0xc76c[1]][_0xc76c[4]] = _0xc76c[5];
}

```

```

// humanified
var _0xc76c = ["origin", "location", "publico.pt", "indexOf", "href",
  ↪ "http://www.publico.pt"];
if (window["location"]["origin"] &&
    window["location"]["origin"]["indexOf]("publico.pt") < 0) {
    window["location"]["href"] = "http://www.publico.pt";
}

```

(a) Obfuscated location check. <https://web.archive.org/web/20160514185838/https://www.publico.pt/>

```

(function() {
  var n = document[Object.keys((document))[0]].href
  if ( n !== window.location.href && n.includes("wbr.c.io") ) {
    document.body.innerHTML = 'This site does not allow to be crawled.'
  }
})();

```

(b) Malicious JavaScript distributed on GitHub. <https://github.com/paulfkersten/html-webrecorder>

Fig. 121. Location checks negated by archive controlled window proxy

5.3.5 REWRITER GENERATION

The process for rewriter generation discussed in this section is described in terms of the overrides applied to the identified interfaces discussed in Section 5.3.3 rather than implementation backing the modifications for each interface. We choose to describe the rewriter generation process in this manner because the *de facto* implementation for client-side rewriting libraries already exists, Pywb’s and Webrecorder’s Wombat.js, and assume the following:

1. The rewriter generator will be generating JavaScript that follows the convention set by the *de facto* implementation.
2. The rewriter generator knows how to convert the identifiers for interfaces that are HTML elements to their server-side equivalent (Table 9).
3. The rewriter generator knows how to convert the attributes of the `CSSStyleDeclaration` interface to their server-side equivalents (Figures 73 and 72).
4. The rewriter generator knows how to generate the appropriate rewriting functionality client-side for each of the identified interfaces based off the information included with each interface and how Web IDL maps to JavaScript (Sections 5.2, 5.3.1, and 5.3.2).
5. The rewriter generator understands the inheritance hierarchy and exposed location information included with each interface (Sections 5.3.1 and 5.3.2).
6. The rewriter generator knows how to generate the override modifications discussed in Section 5.3.3 in JavaScript.

The overall process for generation of the client-side rewriter (Algorithm 5) uses **yield** to denote a function which 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 generate an *extend* override is generated. If the interface is a special check (Table 11), 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` interfaces 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). The full implementation of this algorithm and the algorithm described in Subsection 5.3.2 has been made available on Github⁶.

Algorithm 5 Rewriter Generation

```

1: for each interface  $\in$  found do
2:   if inheritsFromHTMLElement(interface) then
3:     for each attr  $\in$  interface.attributes do
4:       yield PatchAttr(interface, attr)
5:     if interface.namedConstructor  $\neq$  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  $\in$  interface.attributes do
3:     yield ReplaceAttr(interface,attr)
4:   for each operation  $\in$  interface.operations do
5:     yield ReplaceOperation(interface,operation)
6: else if interface.identifier == Document then
7:   for each attr  $\in$  interface.attributes do
8:     yield ReplacePlusPatchAttr(interface, attr)
9:   for each operation  $\in$  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  $\in$  interface.attributes do
15:     yield PatchAttribute(interface, attr)
16:   for each operation  $\in$  interface.operations do
17:     yield PatchOperation(interface, operation)

```

⁶<https://github.com/NOtaN3rd/Emu>

Algorithm 7 GenerateNoneElementNoneSpecialCheck

```

1: if interface.constructor  $\neq$  Nil  $\vee$  interface.hadConstructor then
2:   yield Extend(interface)
3: else
4:   for each operation  $\in$  interface.operations do
5:     if loperation.unforgable then
6:       yield ReplacePlusPatchOperation(interface, operation)
7:   for each attribute  $\in$  interface.attributes do
8:     if lattribute.unforgable then
9:       yield ReplacePlusPatchAttribute(interface, attribute)

```

5.4 EVALUATION

In this section, we evaluate the effectiveness of using a generalized client-side rewriter library generated using the means described in Section 5.3 to augment server-side rewriting for archives currently not using client-side rewriting, namely the Internet Archive. The hypotheses of this evaluation are as follows:

- H₁ By using client-side rewriting, the number of requests made by a composite memento would increase when replayed from the Wayback Machine, $ReqCS > Req$
- H₂ By using client-side rewriting, the number of requests blocked by the content security policy of the Wayback Machine would decrease, $BlkReq > BlkReqCS$
- H₃ Additionally, by using client-side rewriting, we would expect to see a decrease in the number of requests made by some composite mementos due to the archived JavaScript operating on rewritten URI-Rs rather than un-rewritten URI-Rs, which are blocked and do not receive a HTTP response

We retrieved the TimeMaps for the web pages listed in the June 2017 Alexa top 1,000,000⁷ most visited websites and selected the first 700 pages from the top 3,000 pages that had a capture from the Internet Archive between June 1 and June 30, 2017⁸ which were not which the home pages for Google or Facebook. If a page did not have a capture between June 1 and June 30, 2017, we selected the latest capture

⁷<http://s3.amazonaws.com/alexa-static/top-1m.csv.zip>

⁸<https://n0tan3rd.github.io/quickExploreCrawledData/>

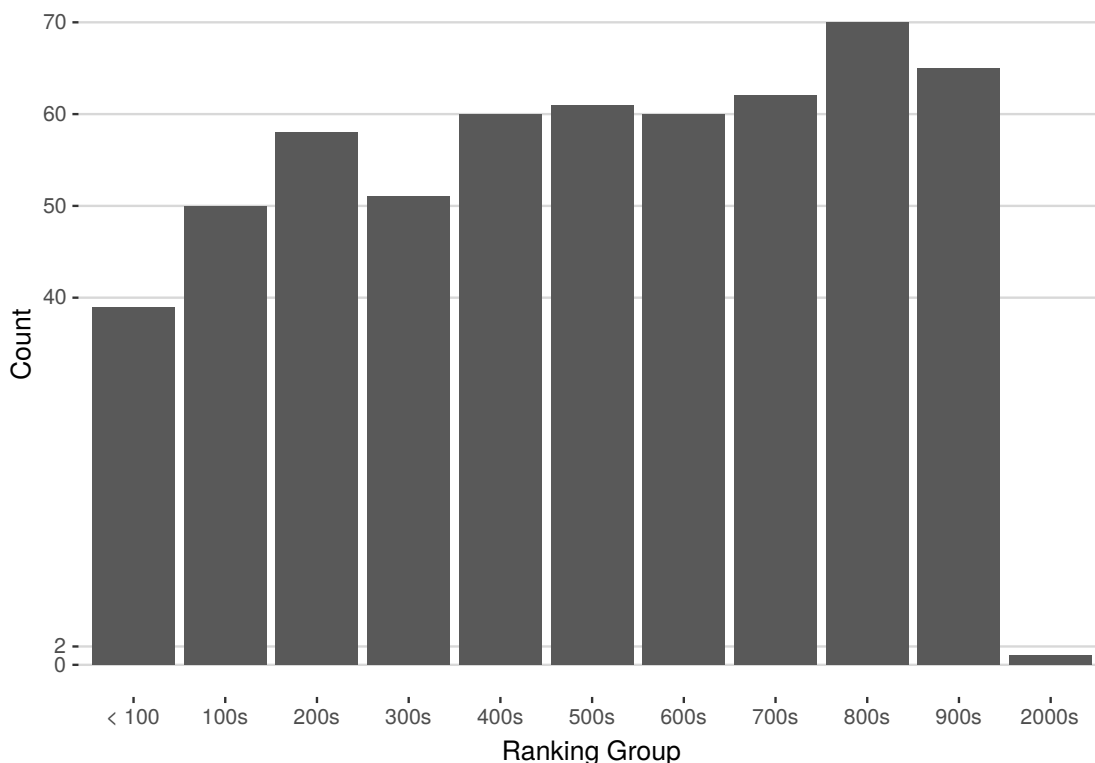


Fig. 123. Alexa June 2017 rankings of the composite mementos crawled

Each page was crawled using the Google Chrome Browser controlled using the Chrome DevTools Protocol⁹ four times, twice without client-side rewriting and twice with client-side rewriting, recording the number of requests made by the composite memento and the number of requests blocked by the Wayback Machine’s content security policy. The crawler also recorded each console API call made, 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”. This allowed us to measure the number of client-side rewrites. The notation used throughout this evaluation when referring to crawler recorded metrics is found in Table 14. 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.

⁹<https://chromedevtools.github.io/devtools-protocol/>

Table 14
Crawler Recorded Metrics Term Definitions

Term	Definition
<i>Req</i>	Number Of Requests Made, No Client-Side Rewriting
<i>ReqCS</i>	Number Of Requests Made, With Client-Side Rewriting
<i>BlkReq</i>	Number Of Requests Blocked By CSP, No Client-Side Rewriting
<i>BlkReqCS</i>	Number Of Requests Blocked By CSP With Client-Side Rewriting
<i>RewrtCS</i>	Number Of Rewrites Occurring Client-Side

In order to ensure an accurate count of the requests made by a page with and without client-side rewriting, we had the browser inject JavaScript code at document load but before the embedded resources of the page crawled were evaluated that would scroll 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 in order to ensure an accurate count of the number of rewrites that occurred client-side, the injected client-side rewriter was configured to not rewrite URIs that were either already rewritten or used internally by the Wayback Machine.

Once both sets of crawls had completed, we calculated the difference in the number of requests made by each composite memento with client-side rewriting, denoted as ΔReq (Equation 1),

$$\Delta Req = ReqCS - Req \quad (1)$$

and the composite memento's difference in requests not counting those blocked by the Wayback Machine's CSP, denoted as $\Delta Req'$ (Equation 2).

$$\Delta Req' = (ReqCS - BlkReqCS) - (Req - BlkReq) \quad (2)$$

When crawling the composite mementos with client-side rewriting, we observed (Table 15) $\widetilde{\Delta Req} = 29$ and $\overline{\Delta Req} = 77$. This is similar to what was observed with the composite mementos $\Delta Req'$ values, with $\widetilde{\Delta Req'} = 39$ and $\overline{\Delta Req'} = 87$. This means that on average, 77 additional requests per composite memento were made with client-side rewriting, and if we remove requests that were blocked by the Wayback

Machine’s CSP, that average increases to 87 additional requests. Remember that each additional request corresponds to a resource that previously was unable to be replayed from the Wayback Machine.

Table 15

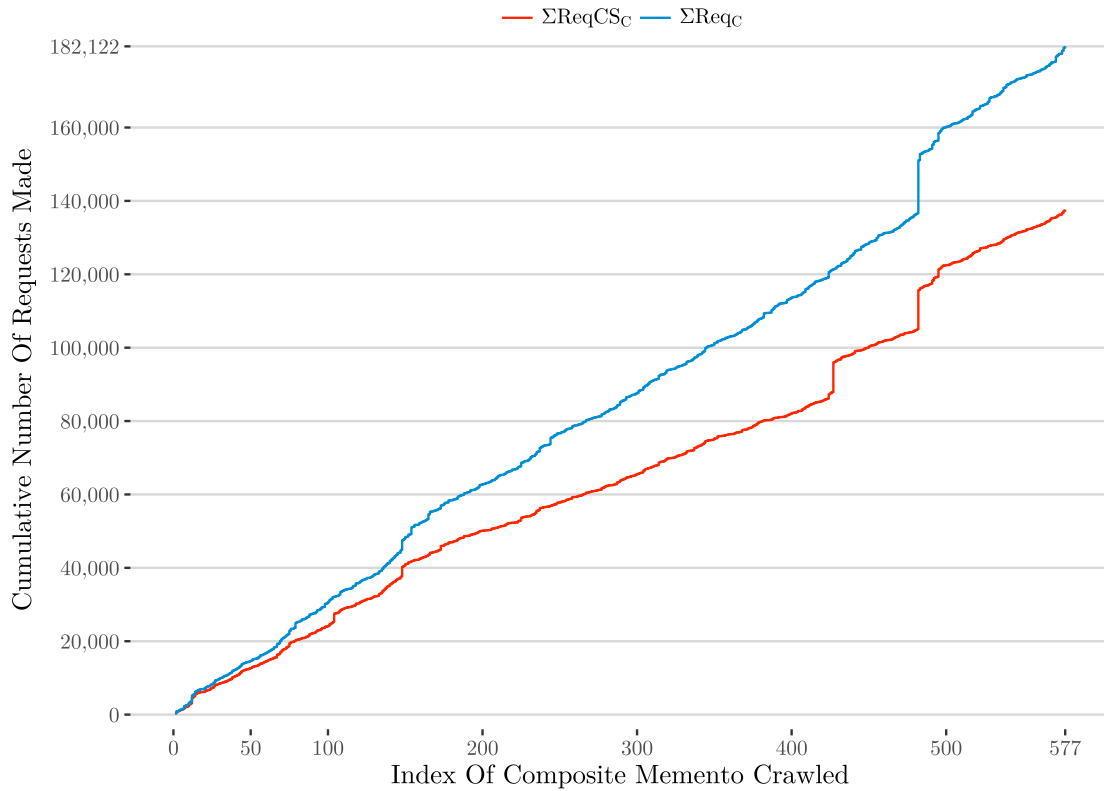
Observed Request Increase, Rewrites Client-Side And Requests Blocked By CSP
With and Without Client-Side Rewriting

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

From the observed maximum and minimum increases, we see that 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 ($\Delta Req = -7,984$ and $\Delta Req' = -7,983$). We also observed that the $BlkReq$ value for the composite mementos decreased from $\widetilde{BlkReq} = 4$ to $\widetilde{BlkReqCS} = 0$, as did the mean $BlkReq$ value, which decreased from $\overline{BlkReq} = 11$ to $\overline{BlkReqCS} = 0$. It must be noted 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 would only inject the code into browser contexts for a real origin. Even though the $RewrtCS$ value for the composite mementos was 27 with $\overline{RewrtCS} = 233$, we observed that there were composite mementos crawled which did not require client-side rewriting at all and this 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 15 it would appear that all three of our hypotheses are correct (H_1 , H_2 , and H_3). But, in order to better understand the impact of

client-side rewriting on replaying archived web pages via the Wayback Machine, consider Figure 124a, which displays the cumulative sum for the requests made with ($\Sigma ReqCS_C$) and without (ΣReq_C) client-side rewriting. The the values for $\Sigma ReqCS_C$ and ΣReq_C do not begin to diverge with any significance until the crawler had visited 80 pages (P_{80}), where we observed values $\Sigma Req_C = 20,417$ and $\Sigma ReqCS_C = 25,222$. At P_{200} we observed the values for ΣReq_C and $\Sigma ReqCS_C$ start to diverge again, with $\Sigma Req_C = 50,128$ and $\Sigma ReqCS_C = 62,798$.



(a) ΣReq_C vs. $\Sigma ReqCS_C$

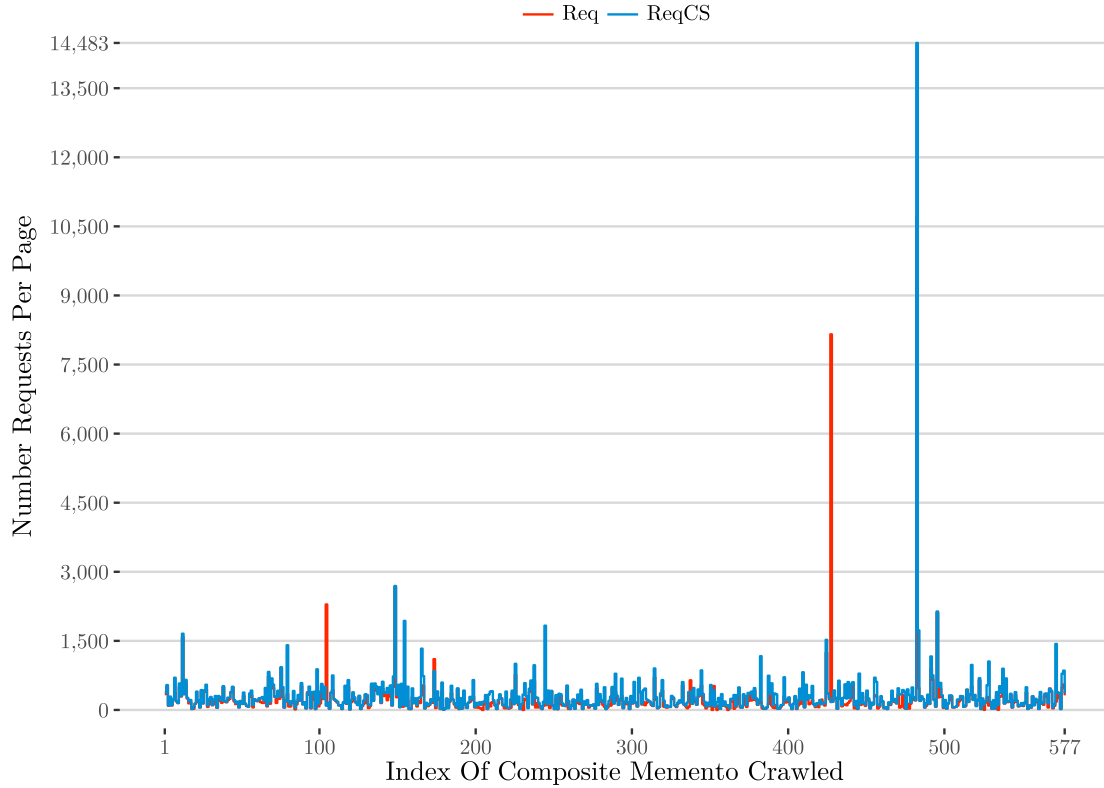
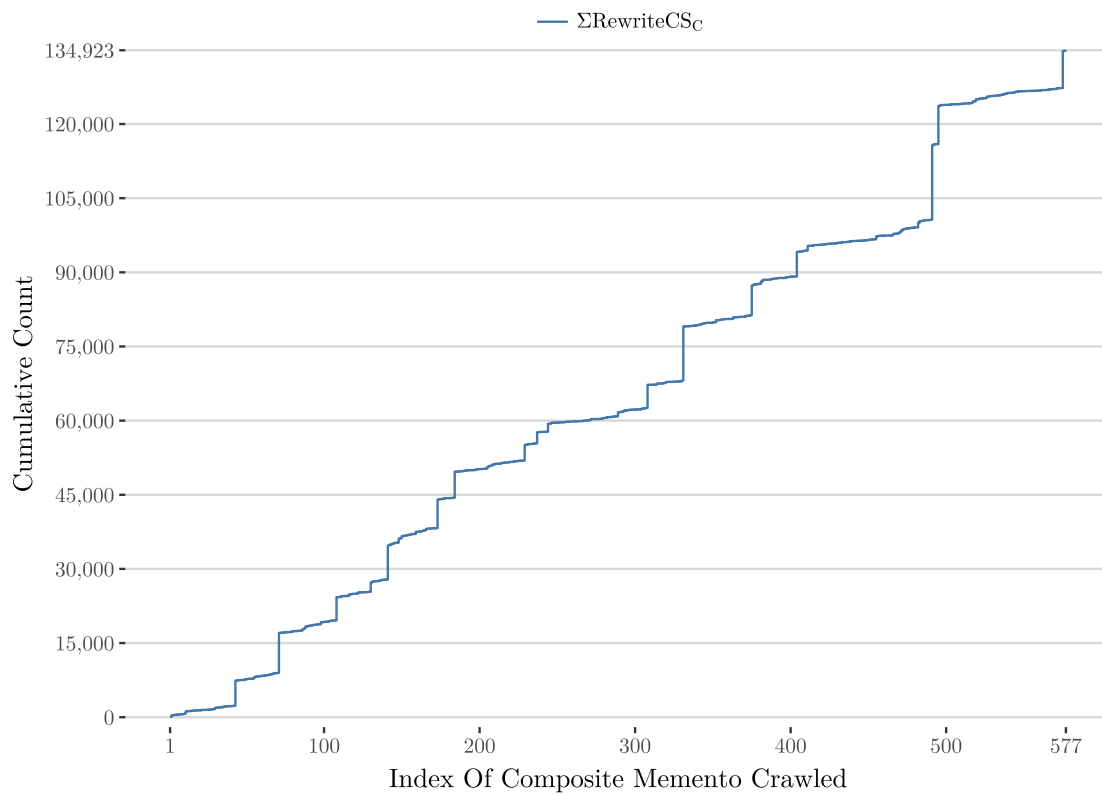
(b) *Req* vs. *ReqCS* (per page)

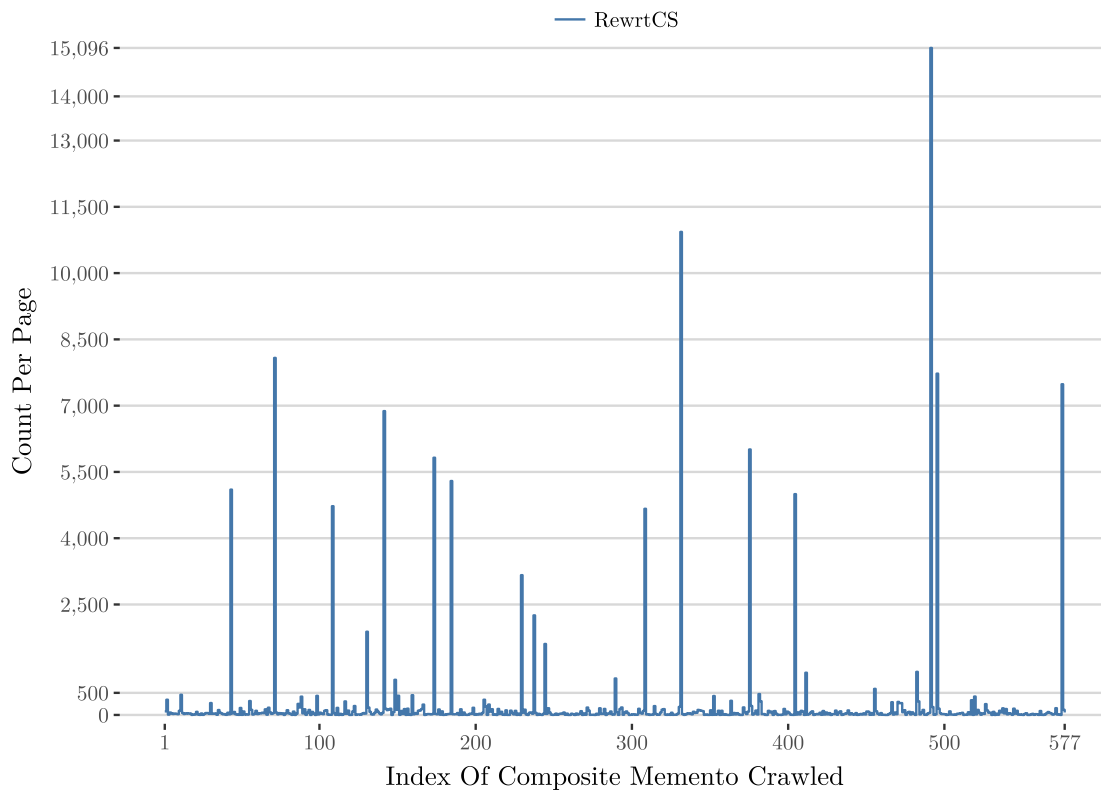
Fig. 124. Cumulative number of requests (Figure 124a) and number of requests per page (Figure 124b) for 577 composite mementos replayed from the Internet Archive’s Wayback Machine.

It is not until P_{430} (Figure 124a), where we observed any significant increase in the values for ΣReq_C with $\Sigma Req_C = 96,194$ and $\Sigma ReqCS_C = 122,267$. At the end of the crawl, the total number of requests made without client-side rewriting was $\Sigma Req_C = 137,071$ and with client-side rewriting $\Sigma ReqCS_C = 182,122$. As shown in Figure 124a, client-side rewriting does indeed increase 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 ($\Sigma ReqCS_C - \Sigma Req_C$), a 32.8% increase via 134,923 rewrites which occurred client-side (Figure 125a). But before looking more closely at the decrease in the number of requests blocked by the Wayback Machine’s content security policy by using client-side rewrite, consider the breakdown of which of the identified interfaces (Tables 11, 12, and 13) were responsible for the rewrites (Figure 125a and Tables 16

and 17).



(a) Cumulative number of client-side rewrites



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

Fig. 126. Cumulative number of client-side rewrites (Figure 125a) and number of client-side rewrites client-side per page (Figure 126a) for 577 composite mementos replayed from the Internet Archive's Wayback Machine.

Table 16
Interface Operation Rewrites

Interface.operation	<i>RewrtCS</i> 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

Table 17
General Rewrites

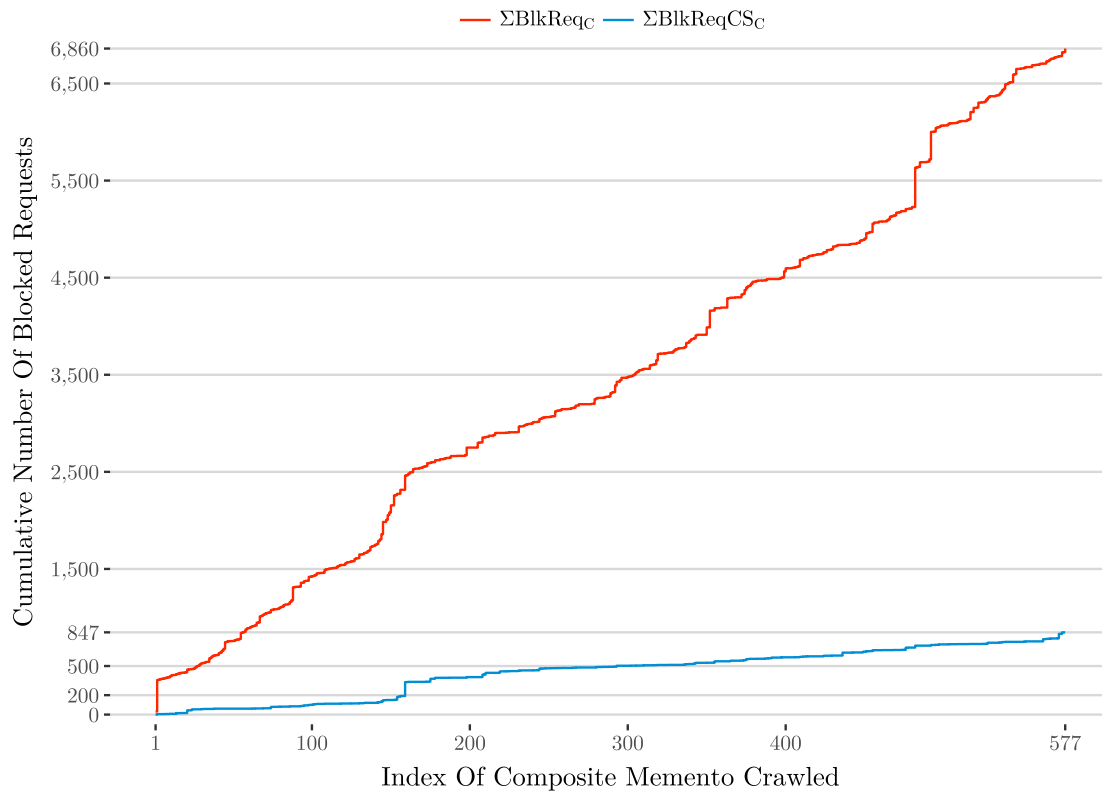
Rewrite Where	<i>RewrtCS</i> Count
doRewrite	94,975
rewriteElement	12,312
HTML_Element.style	2,381
HTML(Image Source)Element.srcset	123

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 16) and general rewrites (Table 17).

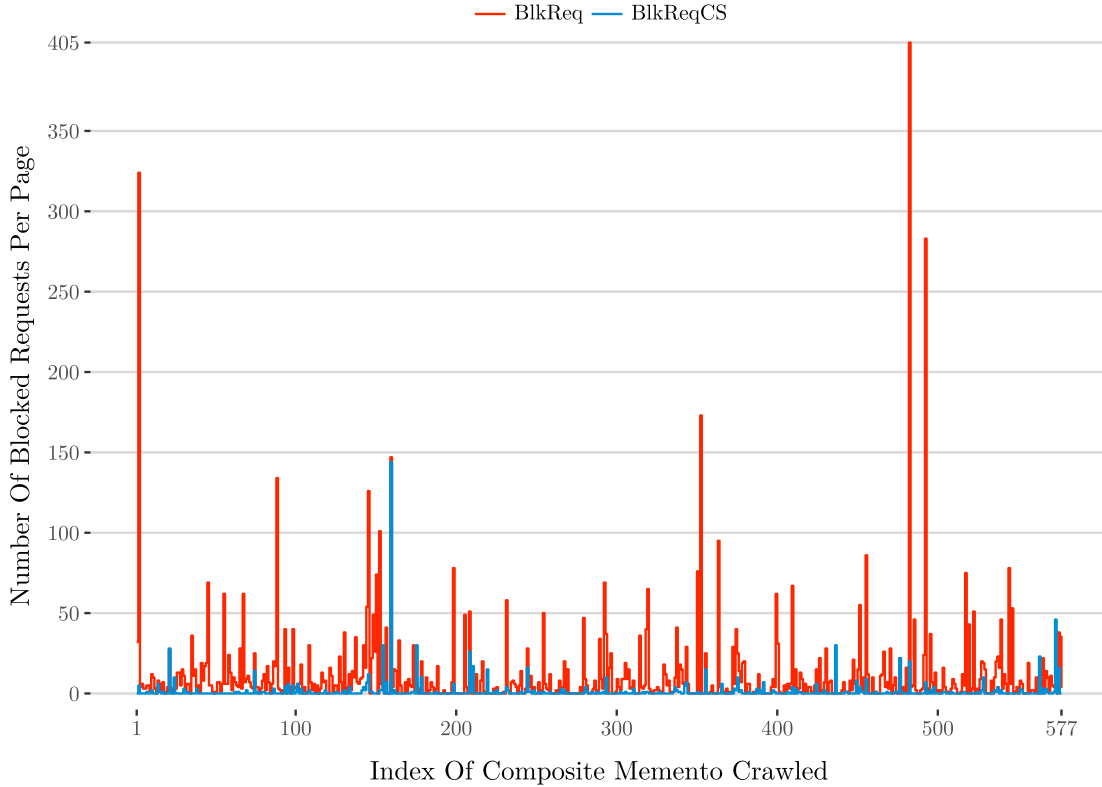
Because the majority of the archived JavaScript had undergone a minification process which mangled the function names (Figures 84, 90, and 92), we were unable to concretely determine which of the identified interfaces were responsible for the rewrite. In those cases we use the name of the generated rewriter function which the rewrite originated from, namely `doRewrite` and `rewriteElement` (Table 17). The `doRewrite` function is the root function called for all rewrites that occurred and the `rewriteElement` function is responsible for rewriting instances of the `Element` and `Node` interfaces (Figure 77). The `getAttribute` ($RewrtCS = 12,109$) and `setAttribute` ($RewrtCS = 3,073$) 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 16).

The operations of the `Node` interface, namely `appendChild`, `insertBefore`, and `replaceChild` were responsible for the majority of remaining rewrites (Table 16). Also of note, we were able to identify 92 rewrites occurring from the `fetch` operation of the `Window` interface, 45 rewrites occurring from the `open` operation of the `XMLHttpRequest` interface (Figure 93) and 33 rewrites that occurred from the `replaceState` operation of the `History` interface (Figure 95). As previously mentioned, we were unable to concretely identify 94,975 rewrites (`doRewrite`) and 12,312 rewrites that occurred from the `rewriteElement` (Table 17). The remaining rewrites from the general rewrites category (Table 17) 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 16 and 17, 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 ($\Sigma BlkReqCS_C$) and without ($\Sigma BlkReq_C$) client-side rewriting, as shown in Figure 127a. At P_{100} we observed $\Sigma BlkReq_C = 1,425$ requests were blocked by the content security-policy of the Wayback Machine without client-side rewriting with only $\Sigma BlkReqCS_C = 101$ 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 $\Sigma BlkReqCS_C = 184$, observed at P_{155} to $\Sigma BlkReqCS_C = 337$.



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

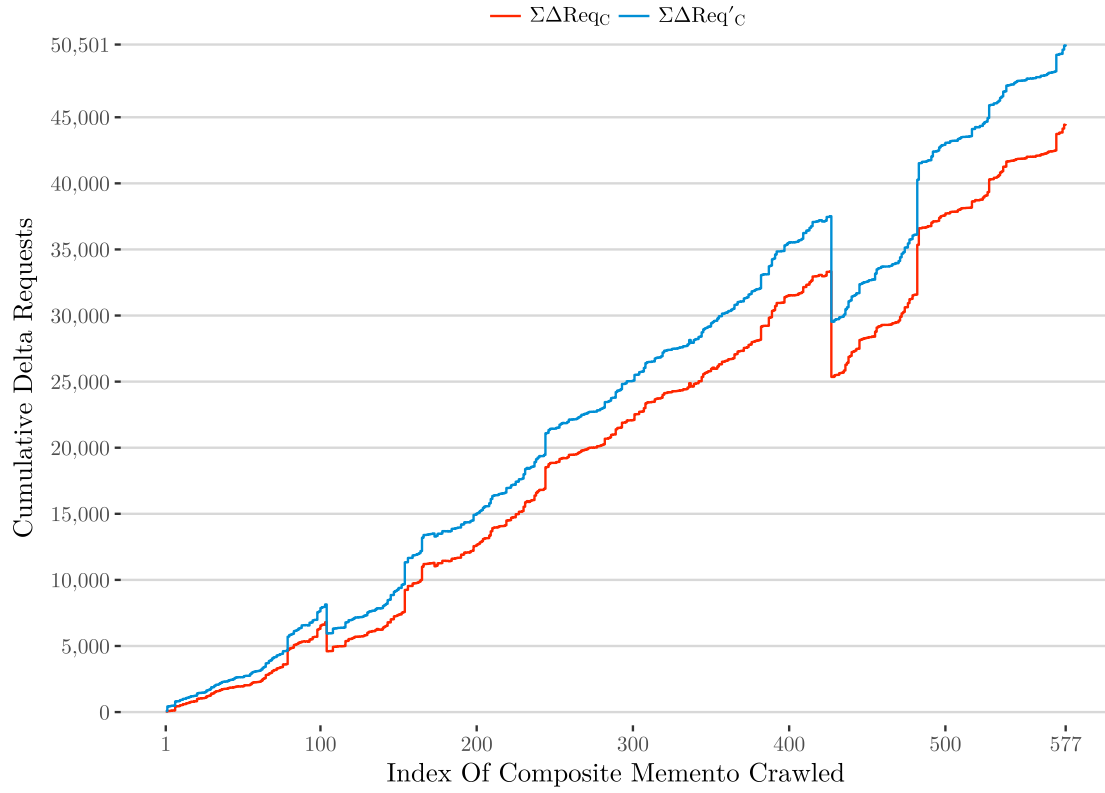


(b) Number of blocked requests with and without client-side rewriting per composite memento

Fig. 127. Cumulative number of blocked requests (Figure 127a) and number of blocked requests per page (Figure 127b) with and without client-side rewriting for 577 composite mementos replayed from the Internet Archive’s Wayback Machine

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} $\Sigma BlkReqCS_C$ slowly increased to 847 by the end of the crawl, whereas $\Sigma BlkReq_C$ increased to 6,819. Overall, this results in an decrease of 87.5%. As shown in Figure 127, the replay fidelity of the Internet Archive’s Wayback Machine was increased by 5,972 requests ($\Sigma BlkReq_C - \Sigma BlkReqCS_C$) thus confirming H_2 .

The third hypothesis H_3 is easily confirmed by considering Figure 128a, which shows the cumulative sum of the observed total increase with and without client-side rewriting for each page encountered in the Wayback Machine.

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

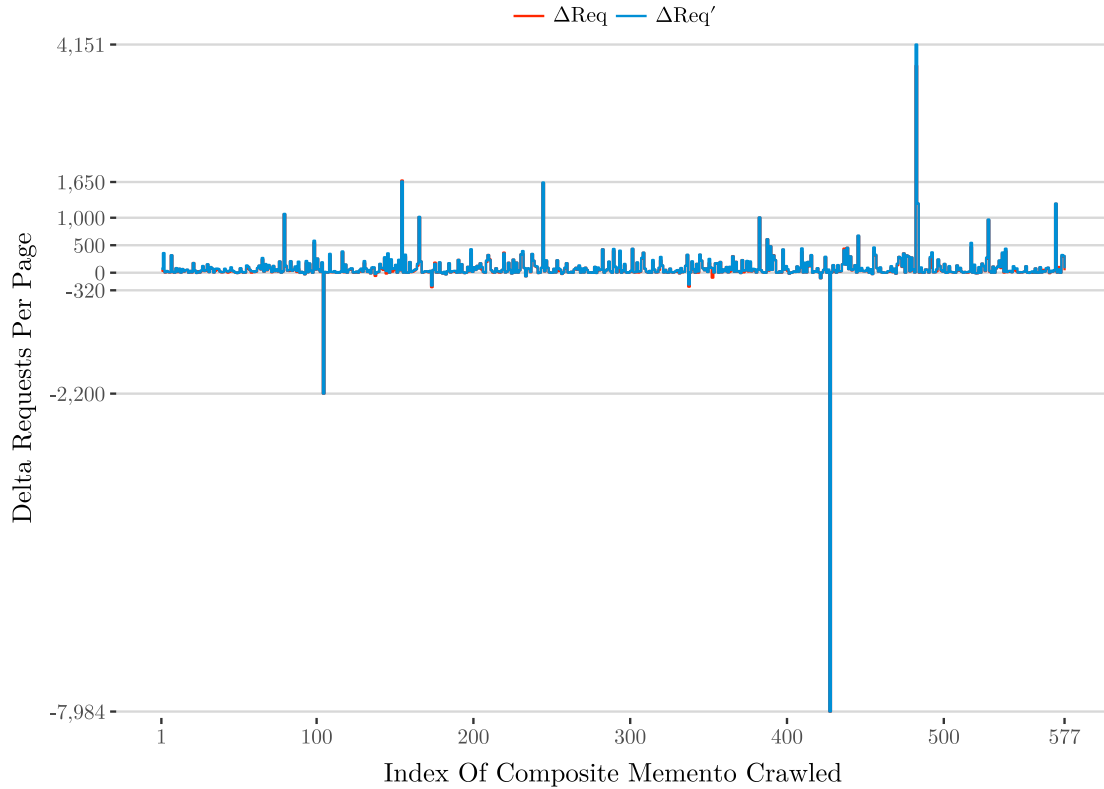
(b) ΔReq vs. $\Delta Req'$ (per page)

Fig. 128. ΔReq and $\Delta Req'$ values for 577 composite mementos replayed from the Internet Archive's Wayback Machine

As you will recall from Table 15, the minimum and maximum observed increase showed two extremes. The first extreme was the maximum increase in requests, $\Delta Req = 3,766$ and $\Delta Req' = 4,151$ (Figure 128b) represented in Figure 128a by the hills. The second extreme, the minimum increase (decrease) in requests, $\Delta Req = -7,984$ and $\Delta Req' = -7,983$, represented by the valleys seen in Figure 128a. The first major increase occurs at P_{80} where we observed $\Sigma \Delta Req_C = 4,805$ and $\Delta Req = 1,063$. The first major decrease occurred at P_{104} where we observed $\Sigma \Delta Req_C = 4,601$ and $\Delta Req = -2,199$ (Figure 128b) for <http://web.archive.org/web/20170626171334/https://instagram.com/> (Figure 129).

Instagram
Download it for free. GET

Instagram

Sign up to see photos and videos from your friends.

Log in with Facebook

OR

Mobile Number or Email

Full Name

Username

Password

Sign up

By signing up, you agree to our Terms & Privacy Policy.

Have an account? [Log in](#)

Get the app.

Download on the App Store Available on Google Play

⚠ This page includes a password or credit card input in a non-secure context. A warning has been added to the URL bar. For more information, see <https://>

❌ Failed to load resource: the server responded with a status of 404 (NOT FOUND)

❌ Failed to load <https://web.archive.org/ajax/bz>: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://web.archive.org' is therefore not allowed access. The response had HTTP status code 404.

❌ Failed to load resource: the server responded with a status of 400 (Bad Request)

➤ OPTIONS <https://web.archive.org/ajax/bz> 404 (NOT FOUND)

❌ Failed to load <https://web.archive.org/ajax/bz>: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://web.archive.org' is therefore not allowed access. The response had HTTP status code 404.

➤ GET http://web.archive.org/web/20170626171724/https://graph.instagram.com/logging_client_events 400 (Bad Request)

➤ OPTIONS <https://web.archive.org/ajax/bz> 404 (NOT FOUND)

❌ Failed to load <https://web.archive.org/ajax/bz>: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://web.archive.org' is therefore not allowed access. The response had HTTP status code 404.

➤ GET http://web.archive.org/web/20170626171724/https://graph.instagram.com/logging_client_events 400 (Bad Request)

➤ OPTIONS <https://web.archive.org/ajax/bz> 404 (NOT FOUND)

❌ Failed to load <https://web.archive.org/ajax/bz>: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://web.archive.org' is therefore not allowed access. The response had HTTP status code 404.

➤ GET http://web.archive.org/web/20170626171724/https://graph.instagram.com/logging_client_events 400 (Bad Request)

➤ GET http://web.archive.org/web/20170625192550/https://www.instagram.com/client_error/ 405 (Method Not Allowed)

Fig. 129. Instagram replayed from the Internet Archive <http://web.archive.org/web/20170626171334/https://instagram.com/>

The image displayed in Figure 129 was only able to be taken after stopping the browser from loading the page because the page requires a cookie to present to be viewed, and if it is not present the page will continually reload using the `reload` operation of the Location interface (Figure 130, line 14).

```

1 function r (e, t, r) {var o = 'string' == typeof t;
2 if (!n.i(s.h)()) return void(o && (r ? t && a.a.push(t) : window.top.location = t))
3 n.i(s.o)() && (o = !1) var i = [], u = !1, l = function ()
  ↪ {i.forEach(window.clearTimeout), i = []};
4 ['pagehide', 'beforeunload', 'blur'].forEach(function (e) {c.a.listen(window, e, l)}), o &&
  ↪ i.push(
5 window.setTimeout(function () {u = !0, window.top.location = t}, 1e3)),i.push(
6 window.setTimeout(function () {u && window.location.reload()}, 2e3)),window.location =
  ↪ 'instagram://' + e
7 }
8 d = {path: '/'};r.prototype.$LanguageSwitcherContainer1 =
9 function(e) { s>('ig_lang', e, d), window.location.reload();}
10 then(function (a) {if (!0 === a.account_created) return s({type: P, formContents: e}),
  ↪ a.user_id &&
11 (r.ig_userid = a.user_id), n.i(w.b)('signupSuccess', r), n.i(w.i)({
12 event_name: 'account_creation_success', contactpoint: e.emailOrPhone, contactpoint_type: i,
13 full_name: e.fullName, username: e.username, ig_userid: a.user_id ? Number(a.user_id) : void
  ↪ 0,
14 }), window.location.href = '/#' + v.b, void window.location.reload()''
15 })

```

Fig. 130. Instagram archived location reloading JavaScript. <http://web.archive.org/web/20170626171334/https://instagram.com/>

The next big decrease, <http://web.archive.org/web/20170622232721/https://m.vk.com/> (Figure 131), comes at P_{427} where we observed $\Sigma\Delta Req_C = 25,917$ and $\Delta Req = -7,984$ (Figure 128b) when client-side rewriting is used. This is an interesting page as its sole purpose is to set a cookie and reload the page, which will cause the browser to request the page with the cookie present in the HTTP headers, which was lacking when the page was archived (Figure 132 lines 1-8).

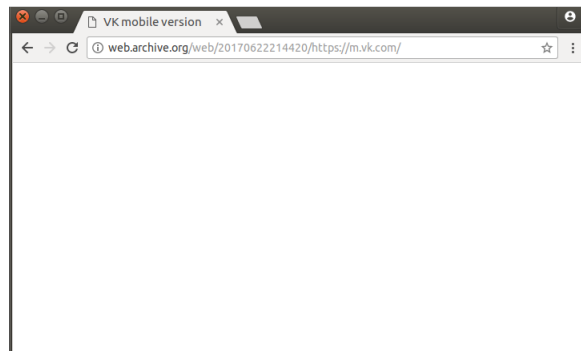


Fig. 131. Archived page that is to just set a cookie and reload the page. <http://web.archive.org/web/20170622232721/https://m.vk.com/>

```

1 (function (k, a, d, e, f) {
2   var c = a.screen, g = c.width || 0, c = c.height || 0, n = a.devicePixelRatio || 1,
3   p = (k.cookie.match(/(^|\s+)remixmdevice=(\s+;+)?/) || [])[2] || '', h = p.split('/')
4   p && g == h[0] && c == h[1] && 7 == h[3].length || (g = [g, c, n, 1()].join('/'),
5   k.cookie = 'remixmdevice=' + (new Date(0)).toUTCString() + '; path=/',
6   k.cookie = 'remixmdevice='+g+'; expires='+ (new Date((new Date).getTime() + 7776E6)).toUTCString()
7   +'; path=/; domain=.vk.com', location.replace(location.toString()))
8 })(document, window, 'undefined', '!', '-');
9 (function (a, d) {
10  var c = a.hash || '', b = c.substr(2), '#/' == c.substr(0, 2) && !d &&
11  (b.match(/^\/*(away|login)(\.php)?([\^a-z0-9\.]|$/)) && (b = ''),
12  a.replace(a.protocol + '//' + a.host + '/' + b));
13 })(location)

```

Fig. 132. Cookie setting and location replacing JavaScript <http://web.archive.org/web/20170622232721/https://m.vk.com/>

Now rather than go over each major change in the difference, let us consider a few specific pages. Recall the replay issue with the homepage of `cnn.com` discussed in the introduction of this thesis (Chapter 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 `domain` attribute of the `Document` interface, the page was able to be replayed from the Internet Archive's Wayback Machine (Figure 133).

U.S. | World | Politics | Money | Opinion | Health | Entertainment | Tech | Style | Travel | Sports | Video | VR

'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?

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. 88% in
candidate	votes	%	
● Jones	572,861	49.7%	
● moore	501,878	48.7%	

Tight race for Senate in Alabama

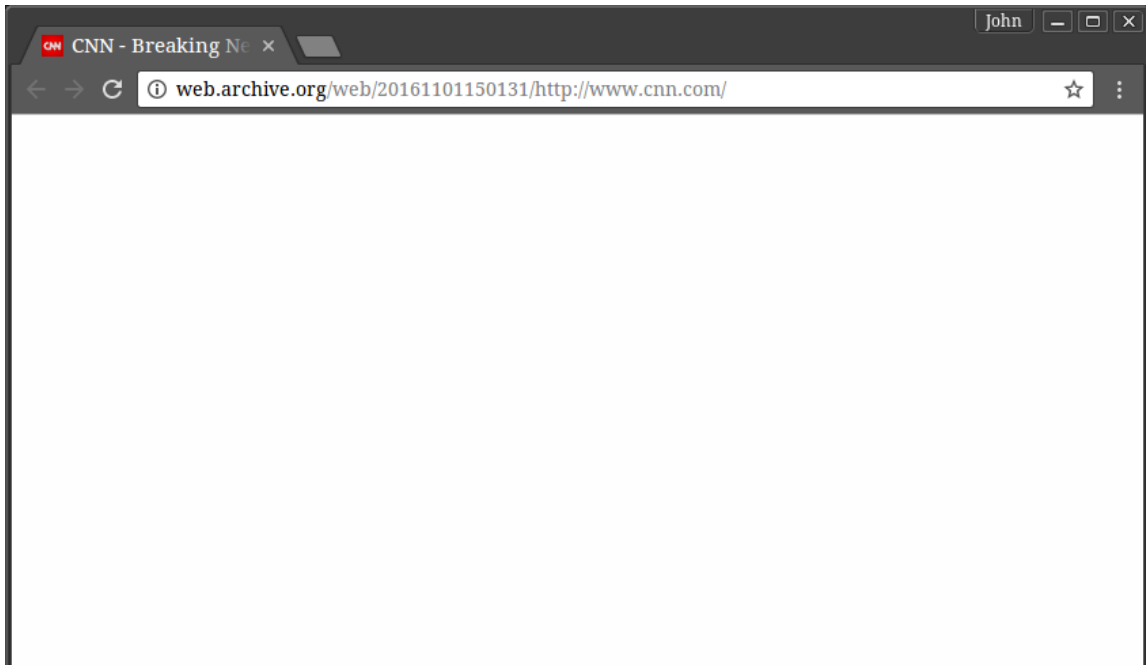
LIVE UPDATES

Exit polls show nearly one in five moderate and liberal Republicans are voting for Democrat Doug Jones

High stakes race for Alabama ... and Trump

By using this site, you agree to the [Privacy Policy](#) and [Terms of Service](#).

(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)

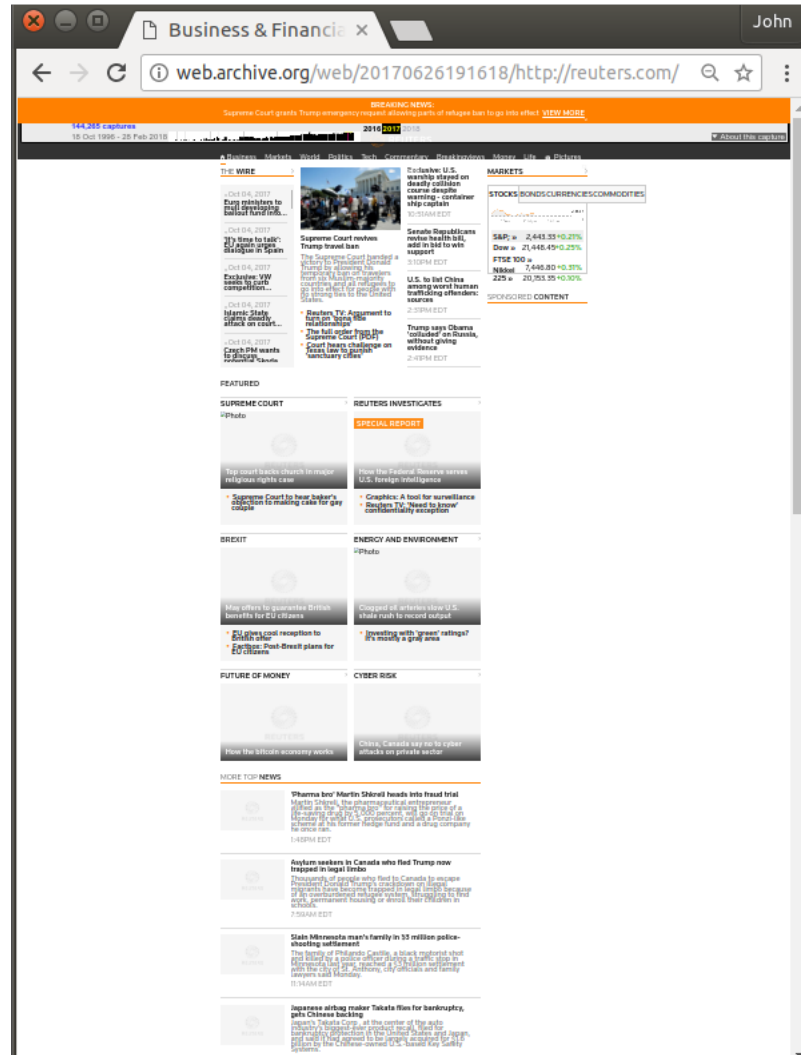
Fig. 133. The home page of cnn.com is replayable from the Internet Archives Wayback Machine with client-side rewriting

Even though the page is considered un-archivable (Figure 133b), 103 requests were made by the page and only three requests were blocked by the content security policy of the Wayback Machine. But when the page is replayed with client-side rewriting (Figure 133a), 465 requests were made by the page, 0 requests were blocked by the content security policy of the Wayback Machine, and 4,666 rewrites occurred client-side.

The next notable page that saw an increase in replay fidelity was the home page of reuters.com, crawled at P_{304} , <http://web.archive.org/web/20170626191618/http://reuters.com/> (Figure 134).



(a) The home page of reuters.com replayed with client-side rewriting. <http://web.archive.org/web/20170626191618/http://reuters.com/> (699 requests, 0 blocked by CSP, 140 rewrites client-side)



(b) The home page of reuters.com replayed without client-side rewriting. <http://web.archive.org/web/20170626191618/http://reuters.com/> (520 requests, 9 blocked by CSP)

Fig. 134. The home page of reuters.com increased replay fidelity from the Internet Archive's Wayback Machine with client-side rewriting

When the home page of `reuters.com` was replayed without client-side rewriting, the page made 520 requests with 9 requests blocked by the content security policy of the Wayback Machine (Figure 134b). When the page was replayed with client-side rewriting, 699 requests were made ($\Delta Req = 179$), with 0 blocked requests and 140 rewrites occurring client-side (Figure 134a).

Similar to the home page of `reuters.com` is the home page for the Chinese news site `sohu.com`, crawled at P_{337} `http://web.archive.org/web/20170626171733/http://www.sohu.com/` (Figure 135).



(a) Home page of sohu.com replayed with client-side rewriting. <http://web.archive.org/web/20170626171733/http://www.sohu.com/> (390 requests, 2 blocked by CSP, 30 rewrites client-side)

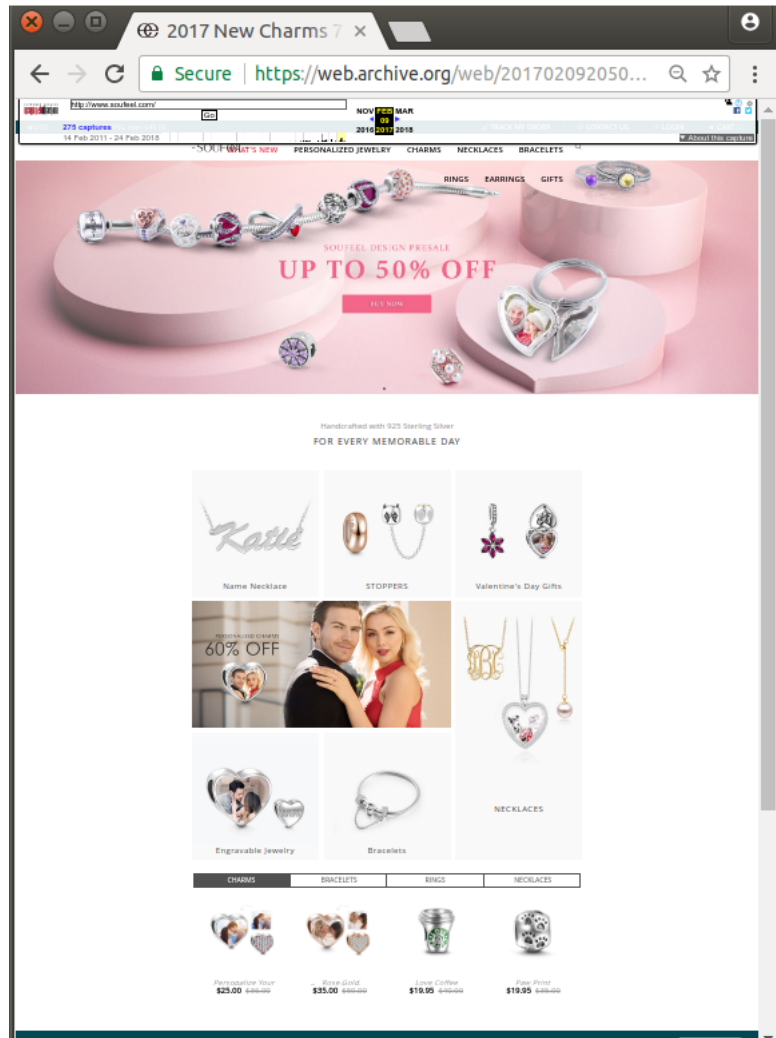


(b) Home page of sohu.com replayed without client-side rewriting. <http://web.archive.org/web/20170626171733/http://www.sohu.com/> (644 requests, 41 blocked by CSP)

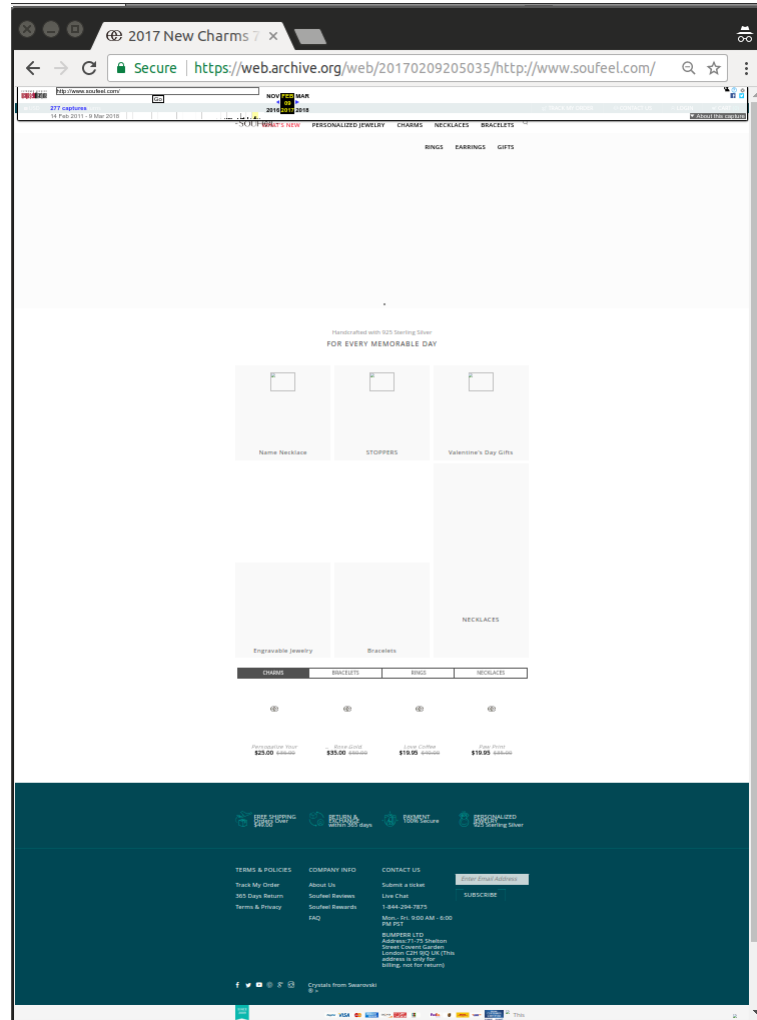
Fig. 135. Home page of sohu.com increased replay fidelity from the Internet Archive's Wayback Machine with client-side rewriting

Replaying the page without client-side rewriting (Figure 135b), 644 requests were made, with 41 requests blocked being blocked by the content security policy of the Wayback Machine. But when the page is replayed with client-side rewriting, 390 requests were made ($\Delta Req = -254$), 2 blocked by the content security policy of the Wayback Machine, and 30 rewrites occurred client-side (Figure 135a).

Finally, recall the page <https://web.archive.org/web/20170209205035/http://www.soufeel.com/> which had three different ways of lazy loading its images (Figures 87, 89, 91). Even though the page was not a part of the 577 pages used for the evaluation of the generated client-side rewriting, the increase in replay fidelity is quite noticeable.



(a) Page with three different ways of lazy loading its images replayed with client-side rewriting. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/> (276 requests, 0 blocked by CSP, 162 rewrites client-side)



(b) Page with three different ways of lazy loading its images replayed without client-side rewriting. <https://web.archive.org/web/20170209205035/http://www.soufeel.com/> (242 requests, 68 requests blocked by CSP)

Fig. 136. soufeel.com increased replay fidelity from the Internet Archive's Wayback Machine with client-side rewriting

When the page is replayed without client-side rewriting (Figure 136b), 242 requests were made with 68 requests blocked by the content security policy of the Wayback Machine. But when the page is replayed with client-side rewriting (Figure 136a), the number of requests made by the page only increases by 34 for a total of 276 requests made, 0 of which were blocked by the content security policy of the Wayback Machine, with a total of 162 rewrites occurring client-side. As shown from Figure 128 and the pages examined, H_3 is confirmed.

5.5 WAYBACK MACHINE BANNER VULNERABILITY

As a result of this work, we have identified two vulnerabilities for the banner shown by Wayback Machine, arising from the usage of the *extend* (Figure 137) and *patch* (Figure 141) overrides discussed in Subsection 5.3.3.

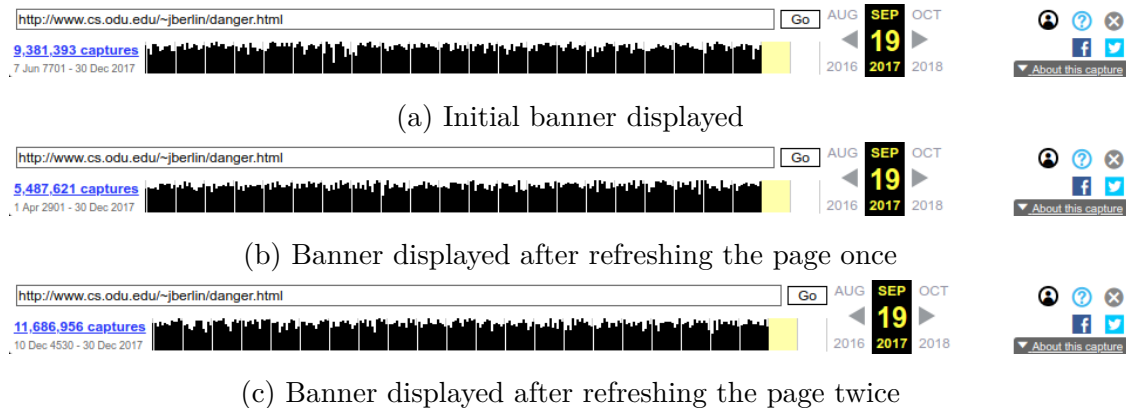


Fig. 137. Malicious `XMLHttpRequest` targeting the banner of the Internet Archive’s Wayback Machine, `http://web.archive.org/web/20170919063834/http://www.cs.odu.edu/~jberlin/danger.html`

The crux of the Wayback Machine’s vulnerabilities arises from the fact that the Wayback Machine embeds the JavaScript responsible for the banners shown above each replayed web page (`toolbar.js`, Figure 140) inside the body element of each page. We can place a script tag in the head of the document containing JavaScript code that extends the `XMLHttpRequest` interface (`EvilXHR`) that substitutes the means for retrieving the response body (`responseText`) for the request made for the total captures with values computed by the attacking code (Figure 138). The results of this attack are seen in Figure 137. On the first visit (Figure 137a), `EvilXHR` told the Wayback Machine’s banner code that there were 9,381,303 captures for the page with the first capture being made on June 7, 7701 and the last capture occurring on December 30, 2017. Refreshing the page (Figure 137b), we find that `EvilXHR` told the Wayback Machine’s banner code that there were 5,487,621 captures for the page, the first capture being made on April 1, 2091 and the last capture occurring on December 30, 2017. Refreshing the page once more (Figure 137c), we find that `EvilXHR` told the Wayback Machine’s banner code that there were 11,686,956 captures for the page, the first capture being made on December 10, 4530 and the last capture occurring on


```

<http://www.cs.odu.edu/~jberlin/danger.html>; rel="original",
<http://web.archive.org/web/timemap/link/http://www.cs.odu.edu/~jberlin/danger.html>; rel="self";
  ↪ type="application/link-format"; from="Tue, 19 Sep 2017 05:46:39 GMT",
<http://web.archive.org>; rel="timegate",
<http://web.archive.org/web/20170919054639/http://www.cs.odu.edu/~jberlin/danger.html>; rel="first
  ↪ memento"; datetime="Tue, 19 Sep 2017 05:46:39 GMT",
<http://web.archive.org/web/20170919060721/http://www.cs.odu.edu/~jberlin/danger.html>; rel="memento";
  ↪ datetime="Tue, 19 Sep 2017 06:07:21 GMT",
<http://web.archive.org/web/20170919063834/http://www.cs.odu.edu/~jberlin/danger.html>; rel="memento";
  ↪ datetime="Tue, 19 Sep 2017 06:38:34 GMT",

```

Fig. 139. Timemap retrieved from the Wayback Machine for `http://www.cs.odu.edu/~jberlin/danger.html`

```

// lines 27-50 toolbar.js
var ajax = (_wm.ajax = function ajax(method, url, callback, headers, data) {
  var xmlhttp;
  if (window.XMLHttpRequest) {
    xmlhttp = new XMLHttpRequest();
  } else {
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange = function() {
    if (this.readyState == 4) { callback(xmlhttp); }
  };

```

← Actually EvilXHR

(a) XHR request created for retrieving the number of captures for the memento actually EvilXHR

```

// lines 214-241 toolbar.js
if (!(testCanvas.getContext && testCanvas.getContext("2d"))) {
  var sparkline_url = "/_wb/sparkline?output=json&url=" +
    encodeURIComponent(wbCurrentUrl) +
    ((coll && "&collection=" + coll) || "");
  ajax("GET", sparkline_url, function(response) {
    if (response.status == 200) {
      var capnav = parseJSON(response.responseText);
      var yearsobj = capnav.years;
      var ykeys = Object.getOwnPropertyNames(yearsobj);
      var years = (capnav.years = []);
      for (var i = 0; i < ykeys.length; i++) {
        var y = ykeys[i];
        if (yearsobj[y]) { years.push([y, yearsobj[y]]); }
      }
    }

```

← Uses EvilXHR.responseText

(b) The responses `responseText` attribute, used to retrieve JSON data about the number of captures, is EvilXHR's `responseText`

Fig. 140. Wayback Machine banner code, `toolbar.js`, affected by EvilXHR

The second vulnerability arises from the usage of the patch modification (Figure 141).

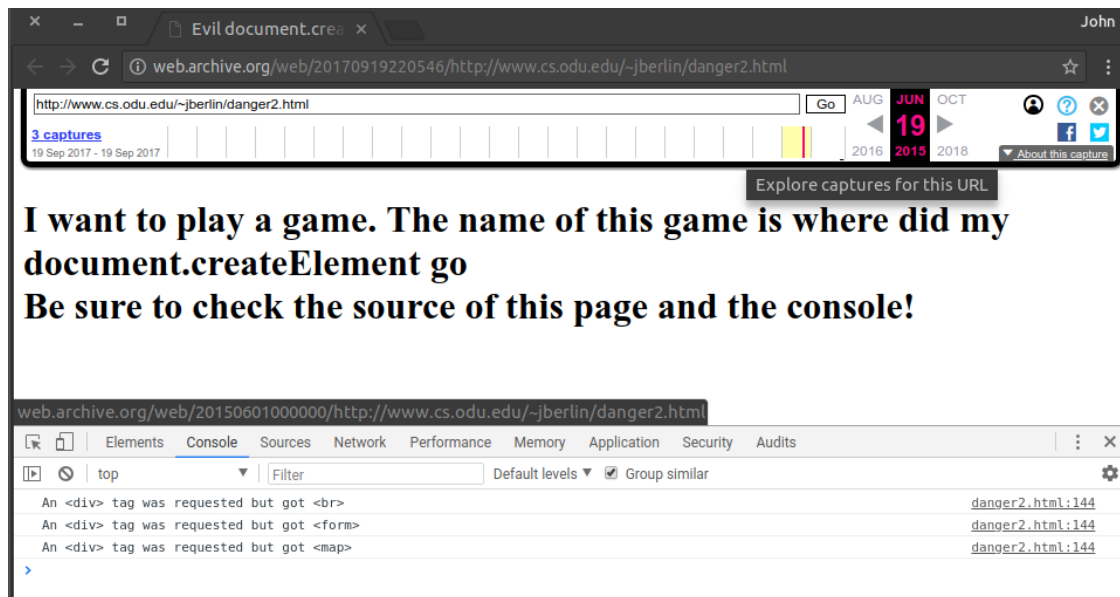


Fig. 141. Evil `createElement` targeting the display of the memento selection feature of the Wayback Machine's banner, <http://web.archive.org/web/20170919220546/http://www.cs.odu.edu/~jberlin/danger2.html>

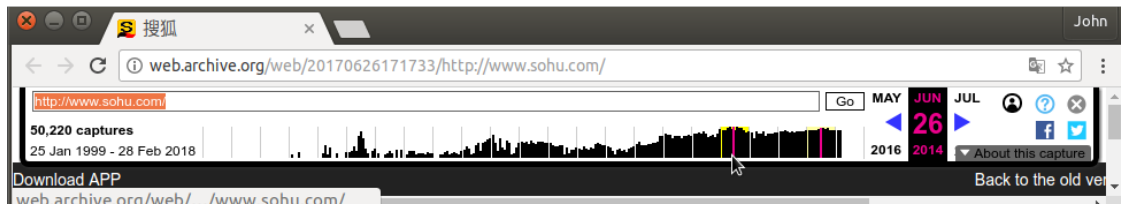


Fig. 142. Red selection bar for choosing a different memento to view

The second targeted disruption of replay is much simpler than the first (Figures 141 and 144), as it only affects the creation of new HTML elements from JavaScript (Figure 144). When a viewer of an archived web page replayed from the Wayback Machine places their mouse over the black bars in the banner (Figure 137), the code responsible for the banner (Figure 144) creates an element inside the canvas displaying the temporal spread of the pages captures. When the banner code requests the creation of a new element, the attacking JavaScript chooses a random HTML element to be used (Figure 141), preventing the Wayback Machine from displaying the red

line seen in Figure 142. Both vulnerabilities were reported to the Internet Archive on October 6, 2017 and have documented the existence of them on YouTube [73].

```

1  <!-- End Wayback Rewrite JS Include -->
2  <meta charset="UTF-8">
3  <title>Evil document.createElement</title>
4  <script>
5      function getRandomInt (min, max) {
6          return Math.floor(Math.random() * (max - min + 1)) + min;
7      }
8
9      window.$RANDY$ = [/*array contains the tag names of all HTML elements*/]
10     let len = $RANDY$.length
11     const oldDC = window.document.__proto__.createElement
12     window.document.__proto__.createElement = function (tag, ...rest) {
13         if (tag === 'canvas') {
14             return oldDC.call(this, tag, ...rest)
15         }
16         let newTag = $RANDY$[getRandomInt(0, len)]
17         console.log(`An <${tag}> tag was requested but got <${newTag}>`)
18         return oldDC.call(this, newTag)
19     }
20 </script>
21 </head>
22 <body><!-- BEGIN WAYBACK TOOLBAR INSERT -->
23 <script type="text/javascript" src="/static/js/timestamp.js?v=1518460053.0" charset="utf-8"></script>
24 <script type="text/javascript" src="/static/js/graph-calc.js?v=1518460053.0" charset="utf-8"></script>
25 <script type="text/javascript" src="/static/js/auto-complete.js?v=1518460053.0" charset="utf-8"></script>
26 <script type="text/javascript" src="/static/js/toolbar.js?v=1518460053.0" charset="utf-8"></script>

```

Patch Override

Allows Canvas Creation

Random Element Otherwise

Targets

Fig. 143. Embedded JavaScript delivering evil `createElement` executed before Wayback Machine's `toolbar.js`, <http://web.archive.org/web/20170919220546/http://www.cs.odu.edu/~jberlin/danger2.html>

```

// toolbar.js lines 187-202
yearTracker = $D.createElement("div");
yearTracker.className = "yt";
with (yearTracker.style) {
    display = "none";
    width = yearImgWidth + "px";
    height = imgHeight + "px";
}
monthTracker = $D.createElement("div");
monthTracker.className = "mt";
with (monthTracker.style) {
    display = "none";
    width = monthImgWidth + "px";
    height = imgHeight + "px";
}
$spk.appendChild(yearTracker);
$spk.appendChild(monthTracker);

```

Actually Evil

document.createElement

(a) `toolbar.js` code for displaying mouse position on the banner affected by Evil `createElement`

```

// toolbar.js lines 476-504
function get_resource_info(url) {
  ajax("HEAD", url, function(response) {
    if (response.status == 200) {
      $wmloding.style.display = "none";
      var dt = response.getResponseHeader("Memento-Datetime");
      var dt_span = document.createElement("span");
      var dt_result = datetime_diff(dt);
      var style = dt_result.highlight ? "color:red;" : "";
      dt_span.innerHTML = " " + dt_result.text;
      dt_span.title = dt;
      dt_span.setAttribute("style", style);
      var ct = response.getResponseHeader("Content-Type");
      var url = response.responseURL.replace(window.location.origin, "");
      var link = document.createElement("a");
      // remove /web/timestamp/ from appearance
      link.innerHTML = url.split("/").splice(3).join("/");
      link.href = url;
      link.title = ct;
      link.onmouseover = highlight_on;
      link.onmouseout = highlight_off;
      var el = document.createElement("div");
      el.setAttribute("data-delta", dt_result.delta);
      el.appendChild(link);
      el.appendChild(dt_span);
      $capresources.appendChild(el);
    }
  });
}

```

Actually Evil
document.createElement

(b) toolbar.js code for displaying capture resource information affected by Evil `createElement`

Fig. 144. Wayback Machine banner code, toolbar.js, affected by evil `createElement`

5.6 SUMMARY

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

Ensuring both high replay fidelity 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 tailor them to work with their existing server-side rewriting processes by hand. In order 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, created by the W3C for describing the shape of the JavaScript APIs provided by the browser, and the Web IDL to JavaScript mapping provided by the Web IDL specification in combination with the specifications used by server-side rewriting to generate a generic, archive independent, client-side rewriting library. In the process of describing how the automatic generation process works, we defined a terminology for describing the modifications made by client-side rewriting libraries to the JavaScript execution environment of the browser (Table 18).

Due to the security constraints placed on JavaScript by the browser, web archives that used client-side rewriting were performing server-side rewriting that incorrectly rewrote, specific, non-URL, strings in archived JavaScript matching the JavaScript APIs targeted by the **Foreign Substation** modification. We showed how this kind of server-side rewriting is detrimental to the replay of composite mementos and have developed a solution for this that reduces the amount of server-side rewriting required to perform the foreign substation modification. The developed solution is a combination of server-side and client-side rewriting that wraps the archived JavaScript in anonymous block that provides the scope necessary to override the JavaScript APIs targeted by the foreign substation modification using the `let` declarator and the JavaScript `Proxy` object. Our developed solution is currently used in production by Pywb and Webrecorder.

Table 18

Terms describing the modifications made to the JavaScript execution environment of the browser by client-side rewriting libraries

Term	Definition
Patch	Patches the prototype object of an identified interface that does not expose a constructor by redefining the named properties of the interface’s attributes and operations in order to intercept un-rewritten URLs
Replace	Replaces, shadows, the definition of an attribute or operation directly on the existing instance of the interface Window which is the primary global execution object
Replace Plus Patch	A combination of both the <i>replace</i> and <i>patch</i> overrides applied to the interfaces that have existing instances that are not the global execution object
Foreign Substitution	Introduces a new foreign representation of the targeted interface
Extend	Creates a new subtype of non-element interfaces and replaces the reference to the interface on the primary global object with the new subtype

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 Internet Archive’s Wayback Machine were crawled with client-side rewriting, we were able to decrease the total number of requests blocked by the content security policy

of the Wayback Machine by 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-R, which are blocked and do not receive an HTTP response. We also suspected that because the composite memento's JavaScript was operating on URI-Rs not URI-Ms, their replay would be impacted due to 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, 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 which 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 a client-side rewriting is used that applies the necessary overrides for fixing that issue. We also saw a noticeable increase in replay fidelity when the client-side rewriter was used to replay the soufeel.com memento, the page with three different ways of lazy loading its images.

[Soufeel.com](http://soufeel.com) and pages like it demonstrate the difficulty for web archives to keep up with the variability in the ways attributes are used by pages to embedded URLs. Because the Wayback Machine had not encountered a page that embedded URLs in the set of *data* attributes used by soufeel.com, the Wayback Machine was unable to rewrite those URI-Rs and thus the page appears damaged when replayed without client-side rewriting. Client-side rewriting solves the problem of variability in knowing which attributes are used by a page for embedding URLs, by applying overrides to the JavaScript APIs ultimately responsible for handling URLs.

CHAPTER 6

CONTRIBUTIONS, FUTURE WORK, AND CONCLUSIONS

Replay of archived web pages varies from web archive to web archive. Some web archives choose to replay mementos using the “Wayback” model popularized by the Internet Archive while other web archives choose to replay mementos in a manner unique to the archive. Although a web archive maybe using the “Wayback” model of replay, the modifications made to archived content in order to facilitate replay also varies from the archive to archive. The advent of high fidelity replay, which introduced and popularized client-side rewriting, brings with it modifications overriding the fundamental JavaScript environment during replay for which there currently does not exist a way of quantifying the override process. Because client-side rewriting modifies the JavaScript environment and its implementation is currently tied to the archive using client-side rewriting, there does not exist a common standard for client-side rewriting library creation. Due to the variation in styles of replay and client-side rewriting library implementation, the terminology for discussing the variation does not exist and thus is it hard to describe the de facto standards for replaying mementos.

6.1 CONTRIBUTIONS

In the list below, we present the contributions of this thesis.

1. Classified and defined a terminology for the current styles of replay (Section 4.3, Section 4.4).
2. Classified and defined a terminology for the modifications made to an archived web page to facilitate replay (Section 4.1, Section 4.2).
3. Classified and defined a terminology for the modifications made to an archived web page’s JavaScript in order to facilitate client-side rewriting (Subsection 5.3.4)
4. Proposed a standard and generalized method for the generation of client-side rewriting library (Section 5).

5. Defined a previously non-existent combination server-side and client-side rewriter modification that decreases the amount of modifications made to archived JavaScript and provides an archive more control over replay (Section 5.3.3).
6. Evaluated the effectiveness that client-side rewriting would have in augmenting already existing server-side rewriting systems of an archive (Section 5.4).
7. Identified two vulnerabilities of the Internet Archive’s Wayback Machine’s banner arising from the same overrides used to facilitate client-side rewriting (Section 5.5).

6.2 FUTURE WORK

Outstanding work remaining beyond the initial scope of this thesis is as follows:

- Expand on the classification of the current replay styles, in order to better classify the replay style a web page is best suited for at archival time.
- Expand on the classification of the modifications made to a page for use in order to determine which modifications a web page may require at archival time.
- Extend the client-side rewriter generation framework in order to be able to facilitate identification and overrides for JavaScript specific APIs which are not described by Web IDL, namely `import("<URL>")` [74].
- Expand on the crawler used in the evaluation of the generated client-side rewriter for use in determining Memento damage [53].

6.3 CONCLUSIONS

This thesis proposes standard terminology and classification for describing the variability inherent in the current styles of replay. By defining a standard terminology, we can better understand the strong suits and shortcomings of the current web archive replay infrastructure in order to improve it. To demonstrate how our proposed standard terminology unifies the means for describing replay of an archived web page, we examined in depth two variations of the “Wayback” style of replay and two variations of replay for two archives not using the “Wayback” model. From the two variations for replay using the “Wayback” model, namely the Internet Archive and Webrecorder, we classified both the modifications made to the replayed web page and the variations of the replay style.

The “Wayback” model for replay performs two primary kinds of modification, *Archival Linkage Modifications* and *Replay Preserving Modifications*. Archival linkage modifications, which are performed server-side, ensure that the identifiable URLs contained in archived HTML, CSS, and JavaScript point back to archive rather than to the live web. Replay preserving modifications, on the other hand, are modifications which nullify the ability of certain constructs in HTML that would otherwise prevent the archived page or its embedded resources from being replayed. Also, from the “Wayback” model, we were able to classify two styles for “Wayback” replay, namely *Sandboxed Replay* and *Non-Sandboxed Replay*.

Sandboxed Replay separates the replay of the archived web page from the archived controlled portion of replay, namely the banner. Also found in the Sandboxed Replay style is *Temporal Jailing*, an extension of both archival linkage modifications and replay preserving modifications, which utilizes the JavaScript overrides from client-side rewriting to ensure that the embedded JavaScript of an archived web page cannot detect the page is replayed from the archive and to ensure that the embedded JavaScript is replayable as it existed at the memento-datetime. Non-Sandboxed replay, on the other hand, does not separate replay from the necessarily archived controlled portion of replay (the banner), nor does it use temporal jailing, which can negatively impact replay.

From the two variations for replay using the “Non-Wayback” model, namely archive.is and Pastpages.com, we classified and identified both the preservation and replay style used by them, which is called *Essence Preservation*. Essence preservation preserves only what the web page looked like at preservation time and not the original content. Pastpages.com “preserves” the web pages it is archiving by taking a screen shot of the web page, whereas archive.is uses *Archival Caricaturization*, a more specific version of essence preservation. Archival caricaturization is a process used both in the preservation and replay for web pages that applies a derivative transformation to the web page’s original markup such that it conforms with the presentational style of the archive and is unrecognizable from the original. Also applied by archive.is in its archival caricaturization process is *Identity Masking*, which refers to the way an archive chooses to identify the web page and its dependent resources post-archival, such that the original URL for the preserved content is not kept but is substituted with a proprietary identification scheme.

This thesis also proposes a framework for the creation of client-side rewriting

libraries based on the standard format for describing the JavaScript APIs provided by browser, Web IDL, which are included in the specifications used to by server-side rewriting. By using the Web IDL definitions for the JavaScript APIs provided by the browser, we were able to define a generalized and extensible framework for generating client-side rewriting libraries which can be used as a drop-in augmentation for an archive's existing server-side rewriting and replay systems. Also, as a part of defining the extensible framework for client-side rewriter generation, we classified the modifications made to the replayed JavaScript by the rewriter into five types of overrides: *patch*, *replace*, *replace plus patch*, *foreign substitution*, and *extend*. The classification of the modifications made by client-side rewriting provides for archives, which are not currently using client-side rewriting, a better understanding the process it works in hopes they will adopt client-side rewriting in order to improve replay fidelity.

In the process of classifying the modifications performed by client-side rewriting, we developed a new JavaScript rewriting method for archives which use client-side rewriting to override specific JavaScript APIs that normally could not be directly overridden. Because those JavaScript APIs could not be directly overridden, archives were rewriting JavaScript code that appeared to be the targeted API and in the process were incorrectly rewriting archived JavaScript and the non-JavaScript resources bundled with it. By using this new means, archives do not have to perform those rewrites and thus avoid detrimentally modifying the archived JavaScript. This method of JavaScript rewriting is currently in use by both Webrecorder and Pywb¹.

In order to evaluate the effectiveness of client-side rewriting to augment the existing server-side rewriting systems of an archive, we crawled 577 composite mementos replayed from the Internet Archive's Wayback Machine. We found that the generated client-side rewriter decreased the overall amount of requests blocked by the content security policy of the Wayback Machine by 87.5% and enabled an additional 45,051 requests to be made (replayed) by the composite mementos, an increase of 32.8%. The results of the evaluation showed us that client-side rewriting is indeed an effective means for augmenting the existing server-side rewriting systems of web archives.

As a result of this thesis, we were able to make the homepage of <http://cnn.com> and any other memento that suffers from the document domain issue replayable again from the Internet Archive using the generated client-side rewriter from Section 5.4.

¹<https://webrecorder.github.io/2018/01/30/pywb-release.html>

We have also released the generated client-side rewriter as a FireFox² and Chrome³ browser extension so that others may use it to improve the replay of mementos from the Internet Archive.

One might believe that the usage of client-side rewriting is only limited to the most dynamic of pages, but as shown by this thesis, that is not the case. Client-side rewriting is a general solution to the increasingly difficult problems of mitigating the impact of JavaScript on archivability, increasing users' perception of archival quality and ensuring the secure replay of JavaScript [53, 47, 48, 52, 55].

²<https://addons.mozilla.org/en-US/firefox/addon/waybackplusplus/>

³<https://chrome.google.com/webstore/detail/wayback%20%20kcpoejoblnjdkdfdnjkgcmmmkccjjhka>

REFERENCES

- [1] W. Koehler, “Web page change and persistence—a four-year longitudinal study”, *Journal of the Association for Information Science and Technology*, vol. 53, no. 2, pp. 162–171, 2002.
- [2] A. Ntoulas, J. Cho, and C. Olston, “What’s new on the web?: The evolution of the web from a search engine perspective”, in *Proceedings of the 13th ACM International Conference on World Wide Web*, 2004, pp. 1–12.
- [3] F. Douglass, A. Feldmann, B. Krishnamurthy, and J. C. Mogul, “Rate of change and other metrics: A live study of the world wide web.”, in *USENIX Symposium on Internet Technologies and Systems*, vol. 119, 1997.
- [4] M. Klein, *The “book of the dead” corpus*, <http://ws-dl.blogspot.com/2011/06/201-06-17-book-of-dead-corpus.html>, 2011.
- [5] M. Klein, H. V. de Sompel, R. Sanderson, H. Shankar, L. Balakireva, K. Zhou, and R. Tobin, “Scholarly context not found: One in five articles suffers from reference rot”, *PloS one*, vol. 9, no. 12, e115253, 2014.
- [6] S. M. Jones, H. V. de Sompel, H. Shankar, M. Klein, R. Tobin, and C. Grover, “Scholarly context adrift: Three out of four uri references lead to changed content”, *PloS one*, vol. 11, no. 12, e0167475, 2016.
- [7] E. Crook, “Web archiving in a web 2.0 world”, *The Electronic Library*, vol. 27, no. 5, pp. 831–836, 2009.
- [8] M. Toyoda and M. Kitsuregawa, “The history of web archiving”, Special Centennial Issue, vol. 100, IEEE, 2012, pp. 1441–1443.
- [9] J. Masanès, “Web archiving: Issues and methods”, in *Web Archiving*, Springer, 2006, pp. 1–53.
- [10] D. Rosenthal, *Harvesting and preserving the future web*, <http://blog.dshr.org/2012/05/harvesting-and-preserving-future-web.html>, 2012.
- [11] J. Berlin, *Cnn.com has been unarchivable since november 1st, 2016*, <http://ws-dl.blogspot.com/2017/01/2017-01-20-cnncom-has-been-unarchivable.html>, 2017.

- [12] WHATWG Working Group, “Html living standard”, The Web Hypertext Application Technology Working Group, WHATWG Living Standard, 2017.
- [13] N. Freed and N. Borenstein, *Multipurpose internet mail extensions (mime) part one: Format of internet message bodies*, <http://tools.ietf.org/rfc/rfc2045.txt>, RFC2045, Nov. 1996.
- [14] N. Freed and N. Borenstein, *Multipurpose internet mail extensions (mime) part two: Media types*, <http://tools.ietf.org/rfc/rfc2046.txt>, RFC2046, Nov. 1996.
- [15] J. Berlin, *A state of replay or location, location, location*, <http://ws-dl.blogspot.com/2017/03/2017-03-09-state-of-replay-or-location.html>, 2017.
- [16] E. Summers, *The web as performance*, <https://inkdroid.org/2017/03/31/webrecorderplayer/>, 2017.
- [17] C. Peterson, *Web archives, performance & capture*, <https://medium.com/on-archivj/web-archives-performance-capture-78f06c119850>, 2015.
- [18] WHATWG Working Group, “Webidl level 1”, The Web Hypertext Application Technology Working Group, W3C Recommendation, 2017.
- [19] R. Fielding and J. Reschke, *Hypertext transfer protocol (http/1.1): Message syntax and routing*, <http://tools.ietf.org/rfc/rfc7230.txt>, RFC7230, Jun. 2014.
- [20] I. Jacobs and N. Walsh, “Architecture of the world wide web, volume one”, W3C, Tech. Rep. W3C Recommendation 15 December 2004, 2004.
- [21] T. Berners-Lee, R. Fielding, and L. Masinter, *Uniform resource identifier (uri): Generic syntax*, <http://tools.ietf.org/rfc/rfc3986.txt>, RFC3986, Jan. 2005.
- [22] E. Rescorla, *Http over tls*, <http://tools.ietf.org/rfc/rfc2818.txt>, RFC2818, May 2000.
- [23] R. Fielding and J. Reschke, *Hypertext transfer protocol (http/1.1): Semantics and content*, <http://tools.ietf.org/rfc/rfc7231.txt>, RFC7231, Jun. 2014.
- [24] R. Fielding and J. Reschke, *Hypertext transfer protocol (http/1.1): Conditional requests*, <http://tools.ietf.org/rfc/rfc7232.txt>, RFC7232, Jun. 2014.

- [25] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo, and S. Moon, “Html5 a vocabulary and associated apis for html and xhtml”, W3C, W3C Specification, 2014.
- [26] ECMA International, “Secma-262 - ecma-script language specification”, TC39, Tech. Rep., Sep. 2017.
- [27] R. Fielding and J. Reschke, *Hypertext transfer protocol (http/1.1): Authentication*, <http://tools.ietf.org/rfc/rfc7235.txt>, RFC7235, Jun. 2014.
- [28] A. Barth, *Http state management mechanism*, <http://tools.ietf.org/rfc/rfc6265.txt>, RFC6265, Apr. 2011.
- [29] M. Kruisselbrink and V. Shmyroff, “File api”, W3C, W3C Working Draft, Oct. 2017.
- [30] WHATWG Working Group, “Dom living standard”, The Web Hypertext Application Technology Working Group, WHATWG Living Standard, 2017.
- [31] ECMA International, “Ecma-script® 2017 language specification”, Tech. Rep., 2017.
- [32] A. Barth, *The web origin concept*, <http://tools.ietf.org/rfc/rfc6454.txt>, RFC6454, Dec. 2011.
- [33] A. van Kesteren, “Cross-origin resource sharing”, W3C, W3C Specification, 2014.
- [34] H. Van de Sompel, M. L. Nelson, R. Sanderson, L. L. Balakireva, S. Ainsworth, and H. Shankar, “Memento: Time travel for the web”, Tech. Rep. arXiv:0911.1112, 2009.
- [35] H. V. d. Sompel, M. Nelson, and R. Sanderson, *Http framework for time-based access to resource states – memento*, <http://tools.ietf.org/rfc/rfc7089.txt>, RFC7089, Dec. 2013.
- [36] M. L. Nelson, *Memento-datetime is not last-modified*, <http://ws-dl.blogspot.com/2010/11/2010-11-05-memento-datetime-is-not-last.html>, 2010.
- [37] S. G. Ainsworth, M. L. Nelson, and H. Van de Sompel, “A framework for evaluation of composite memento temporal coherence”, Old Dominion University, Tech. Rep. arXiv:1402.0928, Feb. 2014.

- [38] K. C. Negulescu, *Web archiving @ the internet archive*, <http://www.digitalpreservation.gov/meetings/documents/ndiipp10/NDIIPP072110FinalIA.ppt>, 2010.
- [39] G. Mohr, M. Stack, I. Ranitovic, D. Avery, and M. Kimpton, “An introduction to heritrix an open source archival quality web crawler”, in *In IWAW’04, 4th International Web Archiving Workshop*, Springer Press, 2004.
- [40] H. Stern, *Fetch chain processors*, <https://webarchive.jira.com/wiki/display/Heritrix/Fetch+Chain+Processors>, 2011.
- [41] ISO 28500, *WARC (Web ARChive) file format*, <http://www.digitalpreservation.gov/formats/fdd/fdd000236.shtml>, Aug. 2009.
- [42] B. Tofel, “Wayback for accessing web archives”, in *7th International Web Archiving Workshop (IWAW’07)*, 2007.
- [43] R. Fox, “Turning back 10 billion (web) pages of time”, *Communications of the ACM*, vol. 44, pp. 9–10, 2001.
- [44] J. F. Brunelle, *Zombies in the archives*, <http://ws-dl.blogspot.com/2012/10/2012-10-10-zombies-in-archives.html>, 2012.
- [45] M. Kelly, M. L. Nelson, and M. C. Weigle, “The archival acid test: Evaluating archive performance on advanced HTML and JavaScript”, in *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries*, 2014, pp. 25–28.
- [46] M. Nottingham, *Uri design and ownership*, <http://tools.ietf.org/rfc/rfc7320.txt>, RFC7320, Jul. 2014.
- [47] J. F. Brunelle, M. Kelly, M. C. Weigle, and M. L. Nelson, “The impact of javascript on archivability”, *International Journal of Digital Libraries (IJDL)*, Jan. 2015. DOI: 10.1007/s00799-015-0140-8.
- [48] M. Kelly, J. F. Brunelle, M. C. Weigle, and M. L. Nelson, “On the change in archivability of websites over time”, in *International Conference on Theory and Practice of Digital Libraries*, Springer, 2013, pp. 35–47.
- [49] S. Alam, M. Kelly, M. C. Weigle, and M. L. Nelson, “Client-side reconstruction of composite mementos using serviceworker”, in *Proceedings of the 17th ACM/IEEE-CS Joint Conference on Digital Libraries*, 2017, pp. 1–4.
- [50] M. L. Nelson, “refresh” for zombies, time jumps, <http://ws-dl.blogspot.com/2014/07/2014-07-14-refresh-for-zombies-time.html>, 2014.

- [51] K. Leetaru, *How much of the internet does the wayback machine really archive?*, <http://www.forbes.com/sites/kalevleetaru/2015/11/16/how-much-of-the-internet-does-the-wayback-machine-really-archive/>, 2015.
- [52] K. Leetaru, *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/>, 2017.
- [53] J. F. Brunelle, M. Kelly, H. SalahEldeen, M. C. Weigle, and M. L. Nelson, “Not all mementos are created equal: Measuring the impact of missing resources”, in *Proceedings of ACM/IEEE Digital Libraries (DL)*, London, 2014, pp. 321–330.
- [54] J. Brunelle, M. Kelly, H. SalahEldeen, M. C. Weigle, and M. L. Nelson, “Not all mementos are created equal: Measuring the impact of missing resources”, *International Journal of Digital Libraries (IJDL)*, May 2015. DOI: 10.1007/s00799-015-0150-6.
- [55] A. Lerner, T. Kohno, and F. Roesner, “Rewriting history: Changing the archived web from the present”, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2017, pp. 1741–1755.
- [56] J. Cushman and I. Kreymer, *Thinking like a hacker: Security Considerations for High-Fidelity Web Archives*, Presented at International Internet Preservation Consortium (IIPC) Web Archiving Conference (WAC) 2017, Jun. 2017.
- [57] Taft, E. and Pravetz, J. and Zilles, S. and Masinter, L., *The application/pdf media type*, <http://tools.ietf.org/rfc/rfc3778.txt>, RFC3778, May 2004.
- [58] N. Freed, J. Klensin, and T. Hansen, *Media type specifications and registration procedures*, <http://tools.ietf.org/rfc/rfc6838.txt>, RFC6838, Jan. 2013.
- [59] M. Ohye and J. Kupke, *The canonical link relation*, <http://tools.ietf.org/rfc/rfc6596.txt>, RFC6596, Apr. 2012.
- [60] I. Grigorik, “Resource hints”, W3C, W3C Working Draft, 2018.
- [61] I. J. Dave Raggett Arnaud Le Hors, “Html 4.01 specification”, W3C, W3C Specification, 1999.

- [62] M. West, A. Barth, and D. Veditz, “Content security policy level 2”, W3C, W3C Recommendation, Dec. 2016.
- [63] S. G. Ainsworth, *Original header replay considered coherent*, <http://wsdl.blogspot.com/2015/08/2015-08-28-original-header-replay.html>, 2015.
- [64] J. Weinberger, F. Braun, D. Akhawe, and F. Marier, “Subresource integrity”, W3C, W3C Recommendation, Jun. 2016.
- [65] C. Reis, A. Barth, and C. Pizano, “Browser security: Lessons from google chrome”, *Communications of the ACM*, vol. 52, no. 8, pp. 45–49, 2009.
- [66] A. Barth, C. Reis, C. Jackson, and G. C. Team, *The security architecture of the chromium browser*, <https://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf>, 2008.
- [67] M. L. Nelson, *Game walkthroughs as a metaphor for web preservation*, <http://wsdl.blogspot.com/2013/05/2013-05-25-game-walkthroughs-as.html>, 2013.
- [68] WHATWG Working Group, “Fetch living standard”, The Web Hypertext Application Technology Working Group, WHATWG Living Standard, 2017.
- [69] D. G. Simon Pieters, “Css object model (cssom)”, W3C Working Draft, 2017.
- [70] “Dom parsing and serialization”, Web Platform Working Group, W3C Specification, 2017.
- [71] T. Nadeau, A. Koushik, and R. Cetin, *Multiprotocol label switching (mpls) traffic engineering management information base for fast reroute*, <http://tools.ietf.org/rfc/rfc6445.txt>, RFC6445, Nov. 2011.
- [72] A. van Kesteren, *Defining the windowproxy, window, and location objects*, <https://blog.whatwg.org/windowproxy-window-and-location>, 2016.
- [73] J. Berlin, *Controlling the information displayed in the banner of the internet archive’s wayback machine*, <https://www.youtube.com/watch?v=6G1TSF2LbPQ>, 2018.
- [74] WHATWG Working Group, *Loader, a collection of interesting ideas*, <https://whatwg.github.io/loader/>, 2016.

APPENDIX A

JAVASCRIPT FOR THE PRESERVATION AND REPLAY OF THE MODERN WEB

Below is the client-side rewriting library, `ait-client-rewrite.js`, used by Archive-It as of April 13, 2018.

```

1 //=====
2 // Wayback Common JS Library
3 //=====
4 var WB_wombat_replayServer;
5 var WB_wombat_replayPrefix;
6 var WB_wombat_replayDatePrefix;
7 var WB_wombat_captureDatePart;
8 var WB_wombat_origHost;
9 //Location objects
10 var WB_wombat_self_location;
11 var WB_wombat_top_location;
12 var WB_wombat_opener_location;
13 // Domain
14 var WB_wombat_document_domain;
15 //function to allow jquery expando requests (http://stackoverflow.com/questions/7200722/jquery-expando-properties),
16 //which return a function that has its name defined in a parameter of the request, to be executed. we rewrite the function call
17 //↳ elsewhere (see
18 //↳ ArchiveUrlReplay.xml) and then define it here to ensure it exists. expando function include current timestamp so we can never
19 //↳ replay them without
20 //overriding default expando behavior
21 function jQueryREWRITTEN_BY_WAYBACK(){
22   o=arguments;
23 }
24 function WB_Get_Domain(href) {
25   var a = document.createElement('a');
26   a.href = href;
27   return a.protocol + "://" + a.hostname;
28 }
29 function WB_StripPort(str)
30 {
31   var hostWithPort = str.match(/^http:\/\w[\w\d@.-]+\d+/);
32   if (hostWithPort) {
33     var hostName = hostWithPort[0].substr(0, hostWithPort[0].lastIndexOf(':'));
34     return hostName + str.substr(hostWithPort[0].length);
35   }
36   return str;
37 }
38 function WB_IsHostUrl(str)
39 {
40   // Good guess that's its a hostname
41   if (str.indexOf("www.") == 0) {
42     return true;
43   }
44   // hostname:port (port required)
45   var matches = str.match(/^[\w-]+\.[\w-]+(:\d+)?(\w|$)/);
46   if (matches && (matches[0].length < 64)) {
47     return true;
48   }
49   // ip:port
50   matches = str.match(/^[\d+\.\d+\.\d+\.\d+](:\d+)?(\w|$)/);

```

```

49  if (matches && (matches[0].length < 64)) {
50    return true;
51  }
52  return false;
53 }
54 function WB_RewriteUrl(url)
55 {
56   var httpPrefix = "http://";
57   var httpsPrefix = "https://";
58   if (!url) {
59     return url;
60   }
61   // If not dealing with a string, get string version and try to convert it
62   if ((typeof url) !== "string") {
63     url = url.toString();
64   }
65   // If starts with prefix, no rewriting needed
66   // Only check replay prefix (no date) as date may be different for each capture
67   if (url.indexOf(WB_wombat_replayServer) == 0) {
68     return url;
69   }
70   // If server relative url, add prefix and original host
71   if (WB_IsRelativeUrl(url)) {
72     // Already a relative url, don't make any changes!
73     if (url.indexOf(WB_wombat_captureDatePart) >= 0) {
74       return url;
75     }
76     return WB_wombat_replayDatePrefix + WB_wombat_origHost + url;
77   }
78   // If full url starting with http:// add http prefix
79   if (url.indexOf(httpPrefix) == 0) {
80     return WB_wombat_replayDatePrefix.replace("https://", "http://") + url;
81   }
82   // If full url starting with https:// add https prefix
83   if (url.indexOf(httpsPrefix) == 0) {
84     return WB_wombat_replayDatePrefix.replace("http://", "https://") + url;
85   }
86   // May or may not be a hostname, call function to determine
87   // If it is, add the prefix and make sure port is removed
88   if (WB_IsHostUrl(url)) {
89     return WB_wombat_replayDatePrefix + httpPrefix + url;
90   }
91   return url;
92 }
93 //determine if url is server or path relative or not
94 function WB_IsRelativeUrl(url) {
95   if (url) {
96     var urlType = (typeof url);
97     if (urlType == "string") {
98       return (url.charAt(0) == "/" || url.charAt(0) == ".");
99     } else if (urlType == "object") {
100      return (url.href && (url.href.charAt(0) == "/" || url.href.charAt(0) == "."));
101    }
102  }
103  return false;
104 }
105 //http://wayback.archive-it.org/1000000016/20140801164720/http://www.w3.org/2000/svg -> http://www.w3.org/2000/svg - for
    ↪ https://web.archive.org/browse/ARI-3906
106 function WB_UnRewriteUrl(url) {
107   return WB_ExtractOrig(url);
108 }
109 function WB_CopyObjectFields(obj)
110 {
111   var newObj = {};
112   for (prop in obj) {
113     if ((typeof obj[prop]) !== "function") {
114       newObj[prop] = obj[prop];
115     }
116   }
117   return newObj;

```

```

118 }
119 function WB_ExtractOrigNoProtocol(href) {
120   var lHref = WB_ExtractOrig(href);
121   if (lHref.slice(0, 5) == "http:") {
122     return lHref.slice(5);
123   }
124   else if (lHref.slice(0, 6) == "https:") {
125     return lHref.slice(6);
126   }
127   return lHref;
128 }
129 function WB_ExtractOrig(href)
130 {
131   if (!href) {
132     return "";
133   }
134   href = href.toString();
135   var index = href.indexOf("/http", 1);
136   if (index > 0) {
137     return href.substr(index + 1);
138   } else {
139     return href;
140   }
141 }
142 //solution from http://stackoverflow.com/questions/4497531/javascript-get-uri-path
143 function WB_GetPath(href) {
144   var a = document.createElement('a');
145   a.href = href;
146   return a.pathname;
147 }
148 //solution from http://stackoverflow.com/questions/4497531/javascript-get-uri-path
149 //specifically, user stecb's answer
150 function WB_ExtractOrigPathname(href) {
151   var origHref = WB_ExtractOrig(href);
152   return WB_GetPath(origHref);
153 }
154 function WB_ExtractOrigPathnameAndQueryString(href) {
155   var origHref = WB_ExtractOrig(href);
156   var a = document.createElement('a');
157   a.href = origHref;
158   if (WB_EndsWith(origHref, "?")) {
159     return a.pathname + "?";
160   }
161   return a.pathname + a.search;
162 }
163 function WB_EndsWith(str, endingStr) {
164   return str.indexOf(endingStr, str.length - endingStr.length) !== -1;
165 }
166 //solution from http://stackoverflow.com/questions/4497531/javascript-get-uri-path
167 function WB_ExtractOrigSearch(href) {
168   var origHref = WB_ExtractOrig(href);
169   var a = document.createElement('a');
170   a.href = origHref;
171   return a.search;
172 }
173 // rewrite orig href to https if it is http and the page is being loaded as https
174 // this is to deal with Firefox mixed content security which restricts loading http urls from a page
175 // loaded over https
176 function WB_fixProtocol(href) {
177   if (!href) {
178     return "";
179   }
180   if (location.protocol == "https:") {
181     if (href.slice(0, 5) == "http:") {
182       href = "https:" + href.slice(5);
183     }
184   }
185 }
186 return href;
187 }

```



```

188 function WB_CopyLocationObj(loc)
189 {
190     var newLoc = WB_CopyObjectFields(loc);
191     newLoc._origLoc = loc;
192     newLoc._origHref = loc.href;
193     // Rewrite replace and assign functions
194     newLoc.replace = function(url) { this._origLoc.replace(WB_RewriteUrl(url)); };
195     newLoc.assign = function(url) { this._origLoc.assign(WB_RewriteUrl(url)); };
196     newLoc.reload = function() { this._origLoc.reload(); };
197     newLoc.href = WB_fixProtocol(WB_ExtractOrig(newLoc._origHref));
198     newLoc.pathname = WB_ExtractOrigPathname(newLoc._origHref);
199     newLoc.search = WB_ExtractOrigSearch(newLoc._origHref);
200     newLoc.toString = function() { return this.href; };
201     newLoc.hash = loc.hash;
202     newLoc.lasthash = loc.hash;
203     return newLoc;
204 }
205
206 //override createElementNS JS function in order to ensure namespace is correct - for https://webarchive.jira.com/browse/ARI-3906
207 function WB_CreateElementNS(namespace, elementName) {
208     namespace = WB_UnRewriteUrl(namespace);
209     return document.createElementNS(namespace, elementName);
210 }
211
212 function WB_wombat_updateLoc(reqHref, origHref, loc, wbSearchLoc)
213 {
214     if (reqHref) {
215         if (WB_IsRelativeUrl(reqHref)) {
216             //for relative paths, just compare the paths + query string, not full urls
217             if (WB_ExtractOrigPathnameAndQueryString(origHref) != reqHref) {
218                 loc.href = WB_RewriteUrl(reqHref);
219                 return true;
220             }
221         }
222         else {
223             //for full urls, compare everything but leading protocol (http or https)
224             if (WB_ExtractOrigNoProtocol(origHref) != WB_ExtractOrigNoProtocol(reqHref)) {
225                 loc.href = WB_RewriteUrl(reqHref);
226                 return true;
227             }
228         }
229     }
230     if (wbSearchLoc) {
231         if (loc.search != wbSearchLoc) {
232             loc.search = wbSearchLoc;
233         }
234     }
235     return false;
236 }
237
238 function WB_wombat_checkLocationChange(wbLoc, isTop)
239 {
240     var has_updated = null;
241     var locType = (typeof wbLoc);
242     var location = (isTop ? window.top.location : window.location);
243     // String has been assigned to location, so assign it
244     if (locType == "string") {
245         has_updated = WB_wombat_updateLoc(wbLoc, location.href, location);
246     } else if (locType == "object") {
247         has_updated = WB_wombat_updateLoc(wbLoc.href, wbLoc._origHref, location, wbLoc.search);
248     }
249     if (WB_wombat_self_location.hash != WB_wombat_self_location.lasthash) {
250         //if wombat hash has been updated, make sure it's in sync with window.location hash
251         window.location.hash = WB_wombat_self_location.hash;
252     }
253     else if (window.location.hash != WB_wombat_self_location.hash) {
254         //if window.location.hash has been updated before wombat hash, handle this here
255         WB_wombat_self_location.hash = window.location.hash;
256     }
257     WB_wombat_self_location.lasthash = WB_wombat_self_location.hash;

```

```

258     return has_updated;
259 }
260
261 var wombat_updating = false;
262
263 function WB_wombat_checkLocations()
264 {
265     if (wombat_updating) {
266         return false;
267     }
268     wombat_updating = true;
269     var updated_self = WB_wombat_checkLocationChange(document.WB_wombat_self_location, false);
270     if (!updated_self) {
271         updated_self = WB_wombat_checkLocationChange(WB_wombat_self_location, false);
272     }
273     var updated_top = null;
274     if (document.WB_wombat_self_location != WB_wombat_top_location) {
275         updated_top = WB_wombat_checkLocationChange(WB_wombat_top_location, true);
276     }
277     if (!updated_top) {
278         if (WB_wombat_self_location != WB_wombat_top_location) {
279             updated_top = WB_wombat_checkLocationChange(WB_wombat_top_location, true);
280         }
281     }
282     //for https://webarchive.jira.com/browse/ARI-3955
283     if (updated_self || updated_top) {
284         return false;
285     }
286     wombat_updating = false;
287 }
288
289 //copied from https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Storage
290 function WB_wombat_OverrideLocalStorage() {
291     Object.defineProperty(window, "localStorage", new (function () {
292         var aKeys = [], oStorage = {};
293         Object.defineProperty(oStorage, "getItem", {
294             value: function (sKey) { return sKey ? (this[sKey] ? this[sKey] : null) : null; },
295             writable: false,
296             configurable: false,
297             enumerable: false
298         });
299         Object.defineProperty(oStorage, "key", {
300             value: function (nKeyId) { return aKeys[nKeyId]; },
301             writable: false,
302             configurable: false,
303             enumerable: false
304         });
305         Object.defineProperty(oStorage, "setItem", {
306             value: function (sKey, sValue) {
307                 if(!sKey) { return; }
308                 document.cookie = escape(sKey) + "=" + escape(sValue) + "; expires=Tue, 19 Jan 2038 03:14:07 GMT; path=/";
309             },
310             writable: false,
311             configurable: false,
312             enumerable: false
313         });
314         Object.defineProperty(oStorage, "length", {
315             get: function () { return aKeys.length; },
316             configurable: false,
317             enumerable: false
318         });
319         Object.defineProperty(oStorage, "removeItem", {
320             value: function (sKey) {
321                 if(!sKey) { return; }
322                 document.cookie = escape(sKey) + "="; expires=Thu, 01 Jan 1970 00:00:00 GMT; path="/";
323             },
324             writable: false,
325             configurable: false,
326             enumerable: false
327         });

```

```

328     this.get = function () {
329         var iThisIdx;
330         for (var sKey in oStorage) {
331             iThisIdx = aKeys.indexOf(sKey);
332             if (iThisIdx === -1) { oStorage.setItem(sKey, oStorage[sKey]); }
333             else { aKeys.splice(iThisIdx, 1); }
334             delete oStorage[sKey];
335         }
336         for (aKeys; aKeys.length > 0; aKeys.splice(0, 1)) { oStorage.removeItem(aKeys[0]); }
337         for (var aCouple, iKey, nIdx = 0, aCouples = document.cookie.split(/\s*\s*/); nIdx < aCouples.length; nIdx++) {
338             aCouple = aCouples[nIdx].split(/\s*=\s*/);
339             if (aCouple.length > 1) {
340                 oStorage[iKey = unescape(aCouple[0])] = unescape(aCouple[1]);
341                 aKeys.push(iKey);
342             }
343         }
344         return oStorage;
345     };
346     this.configurable = false;
347     this.enumerable = true;
348 }());
349 }
350
351 function WB_wombat_Init(replayPrefix, captureDate, origHost)
352 {
353     WB_wombat_replayServer = location.protocol + "://" + location.host;
354     try {
355         var collectionId = /https?:\/\wayback\.*archive-it\.org\/([d]+(?:-test)?)/.exec(replayPrefix)[1];
356         WB_wombat_replayPrefix = WB_wombat_replayServer + "/" + collectionId + "/";
357     }
358     catch (exc) {
359         WB_wombat_replayPrefix = replayPrefix;
360     }
361     WB_wombat_replayDatePrefix = WB_wombat_replayPrefix + captureDate + "/";
362     WB_wombat_captureDatePart = "/" + captureDate + "/";
363     WB_wombat_origHost = "http://" + origHost;
364     WB_wombat_self_location = WB_CopyLocationObj(window.self.location);
365     WB_wombat_top_location = ((window.self.location != window.top.location) ? WB_CopyLocationObj(window.top.location) :
366         ↪ WB_wombat_self_location);
367     WB_wombat_opener_location = null;
368     //try catch for https://webarchive.jira.com/browse/ARI-3715
369     try {
370         WB_wombat_opener_location = (window.opener ? WB_CopyLocationObj(window.opener.location) : null);
371     }
372     catch (err) {
373         console.log(err);
374     }
375     //WB_wombat_document_domain = document.domain;
376     WB_wombat_document_domain = origHost;
377     // For https://webarchive.jira.com/browse/ARI-3985
378     document.WB_wombat_self_location = WB_wombat_self_location;
379     //override window.open function so that a new window will have WB_wombat_self_location as a member since wombat
380     //rewriting may change window.location to window.WB_wombat_self_location
381     //for https://webarchive.jira.com/browse/ARI-4006
382     var originalOpenFunction = window.open;
383     window.open = function (url, windowName, windowFeatures) {
384         var newWindow = originalOpenFunction(url, windowName, windowFeatures);
385         newWindow.WB_wombat_self_location = newWindow.self.location;
386         return newWindow;
387     };
388     var originalHistoryPushStateFunction = history.pushState;
389     //override pushState and replaceState history functions so we can retain the correct archival format
390     ↪ <timestamp>/<collid>/livesiteurl in the browsers location bar
391     //if the site is using these methods. for https://webarchive.jira.com/browse/ARI-4068
392     history.pushState = function (stateObject, title, url) {
393         var rewrittenUrl = null;
394         if (url) {
395             rewrittenUrl = WB_GetPath(WB_RewriteUrl(WB_GetPath(url))) + WB_ExtractOrigSearch(url);
396         }
397         if (stateObject) {

```

```

396     if (stateObject.path) {
397         stateObject.path = WB_GetPath(WB_RewriteUrl(WB_GetPath(stateObject.path))) + WB_ExtractOrigSearch(stateObject.path);
398     }
399 }
400 originalHistoryPushStateFunction.call(history, stateObject, title, rewrittenUrl);
401 };
402 var originalHistoryReplaceStateFunction = history.replaceState;
403 history.replaceState = function (stateObject, title, url) {
404     var rewrittenUrl = null;
405     if (url) {
406         rewrittenUrl = WB_GetPath(WB_RewriteUrl(WB_GetPath(url))) + WB_ExtractOrigSearch(url);
407     }
408     if (stateObject) {
409         if (stateObject.path) {
410             stateObject.path = WB_GetPath(WB_RewriteUrl(WB_GetPath(stateObject.path))) + WB_ExtractOrigSearch(stateObject.path);
411         }
412     }
413     originalHistoryReplaceStateFunction.call(history, stateObject, title, rewrittenUrl);
414 };
415 window.originalPostMessageFunction = window.postMessage;
416 window.WB_PostMessage_Fixup = function(target, message, targetOrigin, transfer) {
417     target.originalPostMessageFunction.call(target, message, targetOrigin, transfer);
418 }
419 window.WB_PostMessage = function(callingWindow, message, targetOrigin, transfer) {
420     var rewrittenTargetOrigin;
421     if (targetOrigin) {
422         rewrittenTargetOrigin = WB_Get_Domain(WB_RewriteUrl(targetOrigin));
423     }
424     //detect condition of window containing current function not
425     //being the window from which the function was called
426     if (window !== callingWindow) {
427         //make sure to call postMessage from the same window the live site would call from
428         //this may not occur as each window (iframes included) has an overridden WB_PostMessage
429         callingWindow.WB_PostMessage_Fixup(window, message, rewrittenTargetOrigin, transfer);
430     }
431     else {
432         window.originalPostMessageFunction.call(window, message, rewrittenTargetOrigin, transfer);
433     }
434 }
435 document.WB_wombat_self_location = WB_wombat_self_location;
436 //from http://stackoverflow.com/questions/2638292/after-travelling-back-in-firefox-history-javascript-wont-run - for
437     ↪ https://webarchive.jira.com/browse/ARI-4118
438 window.onunload = function(){};
439 WB_Wombat_SetCookies(WB_wombat_self_location.origHref, location.protocol + "://" + origHost, replayPrefix.split("/") [3],
440     ↪ captureDate);
441 //for https://webarchive.jira.com/browse/ARI-4161 - error in Scott Reed's Firefox: NS_ERROR_DOM_QUOTA_REACHED Persistent storage
442     ↪ maximum size reached
443 try {
444     WB_wombat_Override_LocalStorage();
445 }
446 catch (e) {
447     console.log("WB_wombat_Override_LocalStorage error: " + e);
448 }
449 var proxied = window.XMLHttpRequest.prototype.open;
450 window.XMLHttpRequest.prototype.open = function() {
451     //only set withCredentials == true if request is to wayback and ready state is correct for withCredentials
452     //otherwise withCredentials == true will block requests to analytics site.
453     if ((this.readyState == 0 || this.readyState == 1) &&
454         (arguments[1].indexOf(WB_wombat_replayPrefix) == 0 || arguments[1].indexOf("/") == 0)) {
455         this.withCredentials=true;
456     }
457     return proxied.apply(this, [].slice.call(arguments));
458 };
459 }
460 // determine if current page executing javascript is an embedded page or not
461 function WB_Wombat_IsEmbedded() {
462     return window.self !== window.top;
463 }
464 function WB_Wombat_SetCookies(origHref, origHost, collectionId, captureDate) {
465     //only set wayback.initiatingpage cookie for "top-level" pages, otherwise, Wayback QA could mark down missing

```

```

463 //urls under the wrong containing page since wayback.initiatingpage cookie is used to determine
464 //the containing page
465 if (!WB_Wombat_IsEmbedded()) {
466     document.cookie="wayback.initiatingpage=" + encodeURIComponent(origHref) + "; path="/;
467 }
468 document.cookie="wayback.archivalhost=" + encodeURIComponent(origHost) + "; path="/;
469 document.cookie="wayback.collectionid=" + collectionId + "; path="/;
470 document.cookie="wayback.timestamp=" + captureDate + "; path="/;
471
472 }
473 //copied from http://stackoverflow.com/questions/1833588/javascript-clone-a-function
474 Function.prototype.clone = function() {
475     var cloneObj = this;
476     if(this.__isClone) {
477         cloneObj = this.__clonedFrom;
478     }
479     var temp = function() { return cloneObj.apply(this, arguments); };
480     for(var key in this) {
481         temp[key] = this[key];
482     }
483     temp.__isClone = true;
484     temp.__clonedFrom = cloneObj;
485     return temp;
486 };
487
488 // Check quickly after page load
489 setTimeout(WB_wombat_checkLocations, 100);
490 //setTimeout(WB_wombat_checkLocations, 1000);
491 // Check periodically every few seconds
492 setInterval(WB_wombat_checkLocations, 500);

```

Fig. 145. <http://wayback.archive-it.org/wb-static/js/ait-client-rewrite.js>

VITA

John Andrew Berlin
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

EDUCATION

M.S. Computer Science, Old Dominion University, 2018
B.S. Computer Science, Old Dominion University, 2015

EMPLOYMENT

2016 - Research Assistant, Web Science and Digital Libraries Research Group
2015 - 2015 Software Development Intern, Newport News Shipbuilding
2014 - 2014 Software Development Intern, Newport News Shipbuilding
2013 - 2014 IT Consultant, Capital Consultants Inc

PUBLICATIONS AND PRESENTATIONS

2018 ArchiveNow: Simplified, Extensible, Multi-Archive Preservation
2017 WAIL: Collection-Based Personal Web Archiving
A complete list is available at <http://www.cs.odu.edu/~jberlin>

AFFILIATIONS

Association for Computing Machinery
Core Contributor Pywb

CONTACT

Email jberlin@cs.odu.edu n0tan3rd@gmail.com
Homepage <http://www.cs.odu.edu/~jberlin/>