The More Things Change, the More They Stay the Same: Integrity of Modern JavaScript

Johnny So
josso@cs.stonybrook.edu
Stony Brook University

Michael Ferdman
mferdman@cs.stonybrook.edu
Stony Brook University

Nick Nikiforakis
nick@cs.stonybrook.edu
Stony Brook University

ABSTRACT

The modern web is a collection of remote resources that are identified by their location and composed of interleaving networks of trust. Supply chain attacks compromise the users of a target domain by leveraging its often large set of trusted third parties who provide resources such as JavaScript. The ubiquity of JavaScript, paired with its ability to execute arbitrary code on client machines, makes this particular web resource an ideal vector for supply chain attacks. Currently, there exists no robust method for users browsing the web to verify that the script content they receive from a third party is the expected content.

In this paper, we present key insights to inform the design of robust integrity mechanisms, derived from our large-scale analyses of the 6M scripts we collected while crawling 44K domains every day for 77 days. We find that scripts that frequently change should be considered first-class citizens in the modern web ecosystem, and that the ways in which scripts change remain constant over time. Furthermore, we present analyses on the use of strict integrity verification (e.g., Subresource Integrity) at the granularity of the script providers themselves, offering a more complete perspective and demonstrating that the use of strict integrity alone cannot provide satisfactory security guarantees. We conclude that it is infeasible for a client to distinguish benign changes from malicious ones without additional, external knowledge, motivating the need for a new protocol to provide clients the necessary context to assess the potential ramifications of script changes.

CCS CONCEPTS

• **Security and privacy → Web protocol security**.

KEYWORDS

JavaScript integrity, web security, measurement

ACM Reference Format:
Johnny So, Michael Ferdman, and Nick Nikiforakis. 2023. The More Things Change, the More They Stay the Same: Integrity of Modern JavaScript. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583395

1 INTRODUCTION

The modern web is a collection of remote resources that are identified by their location. Uniform Resource Locators (URLs) are the standard resource identifiers, specifying the method by which they can be retrieved (protocol), the authority that provides them (domain), and their location within the authority (file path). However, a URL makes no guarantees about the content of the resource at its address, and this is particularly worrisome in the context of browsing the web.

JavaScript is a primary web resource that contains code to be executed in a visitor's browser. A typical web page may require tens of scripts to function as intended, and they are often sourced from remote third parties. Scripts typically provide core functionality (e.g., Ethers [18] to interact the Etherium blockchain) and user experience enhancement (e.g., Bootstrap [4] for styling). Despite the many defenses to prevent rogue script execution, they remain susceptible to supply chain attacks, which inflict harm to an entity by exploiting its trust in less-secure elements. Every additional remote origin that provides JavaScript increases the size of a website's supply chain and, thus, attack surface; unfortunately, websites commonly trust many third parties to provide resources, and these resources are identified by their location with no guarantee of the expected content. Thus, an adversary may be able to compromise the visitors to a domain by compromising one of the domains that are trusted by it to provide its scripts. We continue to witness instances of targeted attacks via providers of third party scripts [25], and some reports measure that half of all cyberattacks today target not only a victim's network, but also its supply chain [3].

No widely-deployed, robust integrity verification mechanism for JavaScript exists on the modern web. The closest such defense, Subresource Integrity (SRI) [30] has seen limited adoption, and the driving force behind the growth it has experienced appears to be copy-paste behavior [5]. The SRI proposal can be considered a *strict integrity* scheme, because it checks for an exact content match by comparing the hash of the content received against the hash of the expected content. However, with the rapid evolution and growth of the web and JavaScript ecosystem, we know that modern scripts frequently change, for a number of reasons. These include: (1) frequent update cycles on the scale of a few days [16], and (2) randomization of select portions of the script content (e.g., comments and variables) [27]. As such, *relaxed integrity* schemes seem better suited in this context. Instead of comparing hashes of expected content, relaxed schemes compare artifacts that describe characteristic attributes of the expected content to tolerate changes within defined bounds.

In this work, we conduct a comprehensive, large-scale study to examine the feasibility of a robust JavaScript integrity defense mechanism and enumerate its requirements. We frame the key goals of this work through the following research questions:

- **RQ0: What is the fraction of scripts that change**? We find that the distributions of scripts along two key dimensions — by their frequency of change in location and change in content — are bimodal, indicating that most scripts change frequently, or not at all. As such, integrity verification schemes must *account for scripts that frequently change, in various dimensions, as first-class citizens.*
- **RQ1: How do scripts change**? We suggest defining script integrity as the integrity of its dimensions, because scripts can change in a number of dimensions beyond their content (e.g., location and context). We empirically discover that *the ways in which individual scripts change remain constant.*
- **RQ2: How applicable is strict integrity verification**? We provide a comprehensive analysis on the infeasibility of strict integrity *at the granularity of the script providers themselves*, by leveraging SRI, the current state of the art, as a running (counter)example. We conclude that *strict integrity alone cannot provide satisfactory integrity guarantees for the web.*
- **RQ3: What are promising directions for future integrity verification mechanisms**? Although there are many dimensions in which scripts can change, individual scripts appear to consistently change in the same manner over time. We recommend *adopting a self-expressive stance regarding web scripts by asking developers to provide metadata of not only the scripts, but also of the ways in which they are expected to change* so that clients can leverage this external knowledge to gain the necessary context to evaluate the ramifications of script changes.

2 BACKGROUND

We assume that a user is able to establish a secure channel with one of the servers of a website and that JavaScript content provided from the remote, third-party providers of this website is potentially malicious. Thus, the user requires some integrity verification mechanism(s) that can check whether the received content is *expected*, and these mechanisms should be able to tolerate changes to the content because modern scripts frequently change, with updates on the scale of a few days [16], and can vary based on uncontrollable parameters [27].

2.1 Integrity Verification

Traditional integrity verification mechanisms, such as the *checksum*, were developed to maintain the integrity of data by detecting unintentional transmission errors. Cryptographic hash functions are used to provide guarantees in the context of an intentionally malicious adversary. The web standard Subresource Integrity (SRI) from the World Wide Web Consortium (W3C) leverages cryptographic hash functions by extending the specification for the HTML script tag with an attribute that specifies the hash value(s) of expected script(s). Then, the user agent (e.g., browser) can compute the hash value of the content that is received, compare it against the specified hash value(s), and load the received script if and only if the hash values match — such an integrity mechanism can be considered *strict*. A *relaxed* mechanism similarly compares artifacts derived from the received content with those derived from the expected content, but the artifacts do not describe the exact content; rather, they describe expected characteristics to tolerate some well-defined level of change. In the context of verifying the integrity of scripts, such artifacts are called signatures or fingerprints, and can be derived from a model such as an Abstract Syntax Tree (AST) that describes the structure of a program.

2.2 The JavaScript Ecosystem

JavaScript comfortably sits as one of the most popular programming languages over time in many reports [11, 12, 29], but there are a number of complications in the JavaScript ecosystem. We summarize them to be: (1) frequent update cycles, with the median lifespan of a script in the order of a few days [16]; (2) script identification, as script URLs can be dynamically generated; (3) dynamic code transformations; and (4) web dynamicity resulting from scripts and conditional content. Script identification is a problem because it is impossible to provide integrity guarantees for individual resources with unpredictable, changing identifiers. Dynamic content modifications can be unpredictable and have been found to apply to the syntax (e.g., variable names or amount of whitespace) and comments (e.g., timestamp) [27]; others might transform data literals of the script (e.g., Google Tag Manager [10]). Thus, clients may receive slightly-altered forms of the same, basic functionality. In terms of web dynamicity, we are mainly concerned with: (1) the user agent (i.e., browser or a programmatic request library), because they support different sets of capabilities (e.g., JavaScript), and (2) bot detection, because servers may present different content if they question the authenticity of the visitor. We address this concern by leveraging "headless" browsers: versions that do not have a graphical display and can be automated.

3 DESIGN & METHODOLOGY

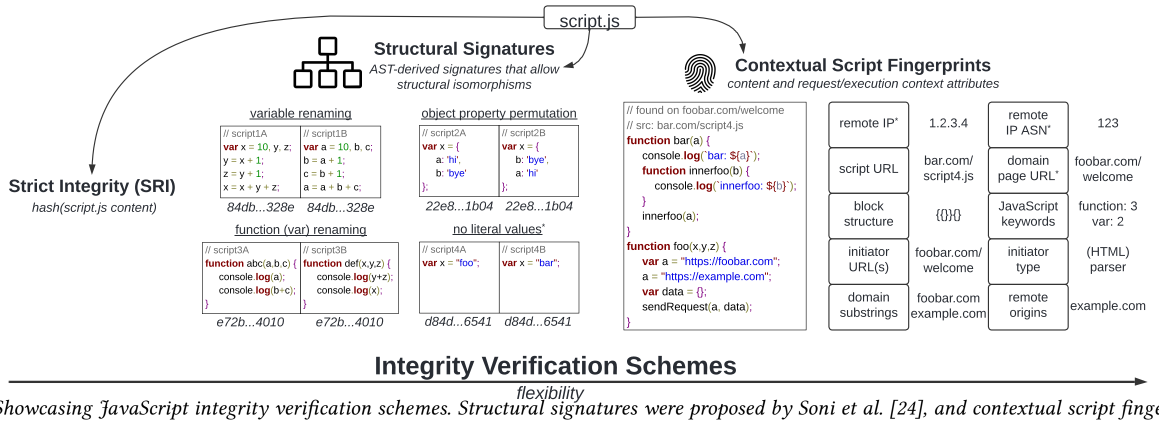
In this section, we present the key methodology behind our script collection and subsequent integrity analyses. We release our code artifacts to the public to encourage future work in this direction [1].

3.1 Script Integrity

As previously mentioned in Section 2, the full script URL may be an unreliable identifier because of dynamic generation. To partially account for this, we use the URL without the query component, and add the requesting domain (i.e., `Referer` of the request) to the script identifier to account for script providers varying their responses by this variable, because renaming and relocating of scripts is a common occurrence, in addition to using variable query strings [6]. In retrospect, this definition still gives rise to an explosion in the number of scripts because of dynamic URLs, as seen in Section 4.

We present an overview of representative JavaScript integrity verification schemes in Figure 1, illustrating SRI, structural signatures proposed by Soni et al. [24], and contextual script fingerprints by Mitropoulos et al. [17]. On one hand, a structural signature is a hash computed over a special type of AST that is designed to permit several isomorphisms — variable renaming, object property permutations, and function (variable) renaming — that represent common, non-structural changes in content. If a structural signature hash changes, that indicates that the functionality of the script has changed through the introduction, removal, or modification of code [24]. However, we note that this *includes changes to data literals in the code, which may seem trivial, but can yield drastic differences in script behavior.* On the other hand, contextual script fingerprints are combinations of various attributes that are extracted from the

[1]https://doi.org/10.5281/zenodo.7192279



```
// script1A
var x = 10, y, z;
y = x + 1;
z = y + 1;
x = x + y + z;
```

```
// script1B
var a = 10, b, c;
b = a + 1;
c = b + 1;
a = a + b + c;
```

```
// script2A
var x = {
    a: 'hi',
    b: 'bye'
};
```

```
// script2B
var x = {
    b: 'bye',
    a: 'hi'
};
```

```
// script3A
function abc(a,b,c) {
    console.log(a);
    console.log(b+c);
}
```

```
// script3B
function def(x,y,z) {
    console.log(y+z);
    console.log(x);
}
```

```
// script4A
var x = "foo";
```

```
// script4B
var x = "bar";
```

```
// found on foobar.com/welcome
// src: bar.com/script4.js
function bar(a) {
    console.log(`bar: ${a}`);
    function innerfoo(b) {
      console.log(`innerfoo: ${b}`);
    }
    innerfoo(a);
}
function foo(x,y,z) {
    var a = "https://foobar.com";
    a = "https://example.com";
    var data = {};
    sendRequest(a, data);
}
```

Figure 1: *Showcasing JavaScript integrity verification schemes. Structural signatures were proposed by Soni et al. [24], and contextual script fingerprints by Mitropoulos et al. [17]. Extra characteristics that we introduce for analysis of script changes are distinguished by a trailing asterisk.*

script content or execution context, and were originally designed to prevent cross-site scripting attacks [16].

Taking a step back, we argue that scripts are resources that not only have highly dynamic execution behavior, but also highly dynamic content that changes. Thus, properly identifying how scripts change must account for how scripts change in a number of dimensions. These include contextual characteristics of the script request (e.g., the request initiator and the `Referer` header), static characteristics of the script content (e.g., raw block structure or information extracted from the AST of a script), dynamic attributes of the script execution (e.g., remote origins contacted for fourth-party scripts), and other miscellaneous characteristics that might be uncontrollable or randomized from the perspective of a client-side integrity verification mechanism. In short, the integrity of a script can be defined as the integrity of its dimensions. We break down the integrity verification proposals into key dimensions and present a taxonomy in Table 1. These integrity dimensions fuel our later analyses in Section 4 to provide insights into how modern scripts change.

3.2 Data Sample

We pick a sample of the top 1 million domains on the Tranco list [14] generated on November 22, 2021 [2] as the set of domains to be crawled daily. To avoid a potential bias in our sample, we accordingly divide the domains into three unevenly-sized buckets to account for the underlying popularity distribution, and we include a random sample of 20,000 domains from each bucket. In particular, the bounds for each rank bucket in interval notation are [1,20K], (20K,500K], (500K,1M]. For each sampled domain, we constrain the set of pages that are crawled every day to ensure that we do not observe false changes in scripts that arise from crawling different parts of a site. To bootstrap the set of pages, our crawlers first send a basic GET request to all domains. If they receive a response within 10 seconds, they will visit the landing page of the domain with a headless browser, attempt to discover up to 10 public pages that are linked from the landing page, and save this set of pages to be crawled every day. The crawlers visit multiple pages of each sampled domain, because a user browsing the web often visits more than just the landing page of a website and other pages of the same site can be drastically different [8, 28]. In retrospect, we found that this choice made it possible for us to find misconfigurations in the already-small set of requests that use SRI, which we describe in Section 5. Domains that failed to reply were retried once every hour for 24 hours; those that failed to respond in this bootstrapping period are permanently excluded.

[2] Available at http://tranco-list.eu/list/9892

Table 1: *A taxonomy of script integrity dimensions, and how scripts change.*

		Dimension	Description
Context	Initiator	Initiator URL	URL of resource that triggered request [17]
		Initiator Type	HTML parser or script [17]
		User Agent	Client type (e.g., browser or request library)
	Location	Script URL (Origin)	Full URL (origin) of the script
		Domain/Page	Domain/URL of the first-party webpage
		IP Address/ASN	Location of the remote provider server
Content	Raw	JS Keywords	# of times JavaScript keywords used [17]
		Block Structure	Braces and parentheses substring [17]
	AST	Domain Substrings	Domain regex matches [17]
		Structural Signature	Hash of special type of AST [24]
Dynamic	Inclusions	Remote Origins	Remote, fourth-party script inclusions [17]
Misc	-	Random Identifiers	Randomized variable names [27]
		Time-varying	Content varies by request time [27]

3.3 Infrastructure

The supporting infrastructure comprises a data collection cluster, distributed database, and analysis pipeline. The data collection cluster consists of three types of workers — the manager node, crawler nodes, and serializer nodes. The manager node determines which sites have yet to be crawled and schedules these, and any necessary retries. After receiving a task, the crawler nodes launch a bare, headless Chromium instance for each domain, and concurrently visit the same set of pages that were initially discovered. After waiting until there are no longer any in-flight network requests, or up to 30 seconds, they discard network request and response chains that did not result in JavaScript, and keep most of the metadata for the remaining network traffic that are made available by the Chrome DevTools Protocol [9]. In order to maximize their efficiency, they send their results to a queue for the serializer nodes to write to the database. The database is distributed across tens of nodes that host Elasticsearch [21]. It is traditionally considered a search engine, but we found that its optimizations for indexing and searching were invaluable to crawling JavaScript at a large scale. The final analysis pipeline consists of a set of notebooks used to generate the presented figures and numbers.

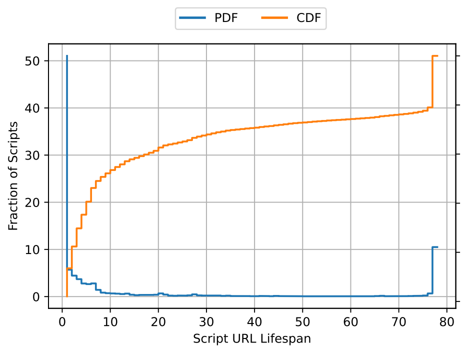


Figure 2: *Bimodal distribution of the URL lifespans of all observed scripts. The left y-axis scale is for the PDF; the right for the CDF.*

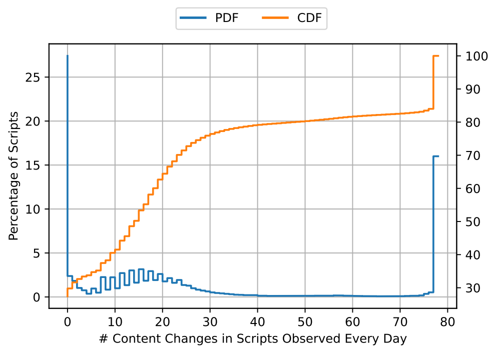


Figure 3: *Bimodal distribution of the number of changes in daily-observed scripts. The left y-axis scale is for the PDF; the right for the CDF.*

Crawling Ethically. The crawlers use automated, headless browsers to visit the same (at most) 10 unique public pages on any given domain per day. These pages were found by traversing links from the landing page of the domain, and the crawlers neither attempt to visit random URLs nor send malformed input of any kind. The requests they send are the same as those sent by a normal user browsing a website. The workers keep a page open for at most 60 seconds, minimizing the number of periodic heartbeat requests. If a page times out, the worker will retry that page once; if the page fails again, the job will be re-enqueued after one hour. Considering the crawl queue size and job completion rate, a completely-unresponsive page will be requested fewer than ten times per day.

When analyzing anomalous data in our initial crawls, we found that some websites detected that the requests were coming from headless browsers. Thus, we attempted to mimic an actual browsing session by instrumenting the headless browsers to appear more similar to the headful version (e.g., mock certain APIs that are not available in headless mode and use a User-Agent value from a version of headful Chrome) [2]. However, we also prioritized transparency with respect to domain administrators. We embed a custom HTTP header (`X-Info`) in all requests, whose value is a URL pointing to a web page that describes the crawling parameters of our experiment, including instructions on how to request exclusion from our daily crawls (we received no such requests during our experiment).

4 MODERN SCRIPT CHANGES

In this section, we begin by describing the collected dataset and then detail the analyses to answer the initial research questions.

4.1 Dataset

Our bootstrapping period started on November 24, 2021 and we present an analysis on the data collected between December 10, 2021 and February 26, 2022 for a duration of 77 days. We choose not to include the first 18 days because we were still refining our data collection infrastructure, but we verified that the exact time window does not affect our results by generating and comparing our results with two disjoint, non-overlapping time windows of two months each. Of the initial 60,000 domains in our sample, we find that: (1) we failed to discover pages for 9,220 domains in our bootstrapping period; (2) we were completely blocked by 321 domains (requests from worker IP addresses would always time out); (3) 2,038 domains did not make any separate script requests (i.e., completely static websites or all scripts are inline); (4) 3,918 domains used entirely first-party scripts; and (5) there remain 44,503 domains whose script requests we analyze.

Using the script definition from Section 2, and excluding first-party script requests and those that were initiated because of an iframe (e.g., an embedded Google Maps iframe), we observe a total of 5,930,962 scripts (with 2,747,077 unique script URLs) from 52,169 unique provider origins, for an average of 133 scripts per domain, or 114 remote script provider origins per domain. On average, our infrastructure ingested 90 GB of data every day (with one replica).

4.2 (RQ0) First-Class Citizens

Prior works reported that frequently-changing scripts comprised a negligible fraction of all scripts and were not used by many domains [24]. This is no longer the case: scripts that frequently change are no longer the exception, but the norm. The distributions of scripts are bimodal along two orthogonal dimensions: (1) by the URL lifespan of a script in Figure 2, and (2) by the number of content changes of a script in Figure 3. In both perspectives, the modes are at opposite extremes, with few scripts that fall in between the two modes, easily lending this behavior to a simple taxonomy based on the dynamicity of its location and of its content. Refer to Table 2 for high level statistics regarding the bimodal distribution of scripts; note that the "Total" buckets account for the scripts that fall in between the two modes.

Distribution of Script URL Lifespans. In Figure 2, the URL lifespan of a script is calculated by taking the difference in days between the last and first dates we observe a script while crawling a domain. We conservatively consider script URLs to be static or dynamic according to the two modes of the distribution: static if they never changed, and dynamic if they changed every day. Scripts with dynamic URLs account for more than half (51%) of the entire set of external scripts in our dataset. We note that this phenomenon occurs even with our script definition that disregards URL query parameters as mentioned in Section 2. Such dynamic-URL scripts typically include an unpredictable identifier in the URL, and a small fraction of domains that contribute an abnormally large amount of such scripts caused an inflation in this particular class of scripts. This is

Table 2: *High-level statistics of **S**tatic and **D**ynamic scripts.*

URL	Content	# Scripts	# Domains	# Origins
D	-	3,027,359 (51%)	27,701 (62%)	21,963 (42%)
S	-	622,620 (11%)	40,281 (91%)	27,680 (53%)
S	S	178,316 (3.0%)	30,279 (68%)	15,341 (29%)
S	D	37,010 (0.6%)	13,870 (31%)	2,088 (4.0%)
Total		5,930,962 (100%)	44,503 (100%)	52,169 (100%)

Table 3: *Variability of daily-changing scripts ($N = 38{,}030$).*

	Same	UA	Time	Indet.
Same	18%	22%	21%	24%
UA	22%	5%	30%	30%
Time	21%	30%	29%	32%
Indet.	24%	30%	32%	1.8%
Totals	47%	41%	68%	39%

in stark contrast to scripts that used the same URL throughout the time window, of which there are 10.5%.

Takeaways: Dynamic-URL scripts are very common in the modern script ecosystem, which makes the task of reliably identifying remote scripts to be much more difficult from an external perspective. Incorporating (some part of) the URL of the script will inevitably cause an explosion in the number of scripts.

Distribution of Script Content Changes. Similarly, from the perspective of the number of content changes as shown in Figure 3, we conservatively categorize scripts as follows: static if they never changed in content, or dynamic if they changed in content every day. Figure 3 considers only the set of scripts that were observed daily (not the set of scripts that were labeled as having static URLs in Figure 2, because those scripts could have disappeared and then reappeared). We encountered 223,515 scripts daily, which is 36% of scripts that were marked as having static URLs. Of these scripts that were observed daily, 27% of them never changed in content, and 16% of them changed in content every day; furthermore, the scripts that changed in content every day were found in 31% of all domains.

Takeaways: A significant portion of scripts that are observed daily also change in content. If an integrity verification mechanism cannot tolerate scripts that change content at least once within two, four, or eight days, it would be inapplicable to 21%, 40%, or 62% of daily-observed scripts, respectively.

4.3 (RQ1) Consistent Patterns of Change

In this subsection, we detail the consistent patterns of change in scripts, and we include complementary analyses in Appendix A.

Dynamic Generation. Table 3 presents a quantified breakdown of how daily-changing scripts vary in content based on the user agent that makes the request and the time of the request, from one day of data. After crawling a page, every script request made by the browser was replayed twice — with the original headers — using a programmatic request library, and then categorized as follows:

(1) `Same` (47%): all three responses are the same.
(2) `UA` (User Agent, 41%): both of the responses to the replayed requests are the same, but different from the original response. These URLs were not one-time URLs, and we theorize this behavior is caused by user agent fingerprinting [1].
(3) `Time` (68%): all three responses are different [27].
(4) `Indet.` (Indeterminate, 39%): the original response is the same as only one of the two responses to the replayed request.

A single script can belong to more than one category, which can occur if it is requested multiple times during one crawl of a domain (e.g., multiple times on the same page or across multiple pages). These statistics suggest that even if daily-changing scripts do not actually receive code updates from their developers, the current strict integrity mechanisms, which only take into account the content of the script, can only be straightforwardly applied to the 18% of them that do not dynamically vary by other parameters. In other words, a significant fraction (82%) of these scripts change because of a number of parameters (including the user agent and time of request) that may not be easily adapted into a strict integrity mechanism such as SRI.

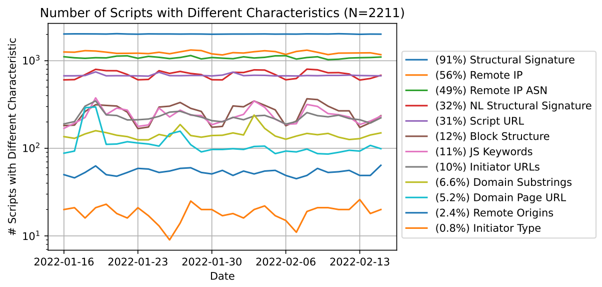


Figure 4: *The number of scripts that have a different characteristic as compared to the previous day; the percentage indicates the mean percentage of change daily.*

Takeaways: A robust mechanism *should* account for scripts that vary based on uncontrollable parameters because they comprise a large portion of daily-changing scripts. Only 18% of such scripts do not dynamically vary, whereas 41% vary by user agent, 68% by time, and 39% by indeterminate parameter(s).

Consistent Types of Change in Scripts. In Figure 4, we present a time-series plot that depicts the number of scripts that experience changes along different dimensions in a 32-day window within our original 77-day analysis window. The plotted sample consists of the first scripts with a new script URL in sorted order, out of the ones that are both frequently-changing and categorized as `Same`, to ensure a reasonable runtime, and to ensure that we compare the same scripts.

We extract Table 1 characteristics from the sample on each day, and check if the values are the same as those from the prior day. A greater number for a particular characteristic implies that a greater number of scripts experience change in that dimension. We highlight two key observations: (1) the number of changes are generally constant; i.e., time does not affect the manner in which these scripts change, and (2) there is a general ranking of attributes that change the most in scripts, with most of the scripts experiencing changes in their structural signature every day. If the structural signature of a script changes, that means it experiences changes that affect script execution or significat changes to code [24]. If we consider an augmented, "no-literal" (NL) version of structural signatures that excludes data literals, the daily mean percentage of changes decreases significantly from 91% to 32%. In other words, an average of 59% scripts experience changes to only data literals, as compared to the 32% that receive changes to code, on a daily basis. Furthermore, we confirmed that these trends persist, even if the sample in Figure 4 had comprised only advertisement and tracking-related scripts as classified by EasyList and EasyPrivacy.

Takeaways: Frequently-changing scripts appear to consistently change in the same dimensions, holding true even for complex, ad-related scripts that appear on filter lists such as EasyList and EasyPrivacy. On average, 59% of such scripts experience changes to only data literals, as compared to the 32% that receive changes to code, on a daily basis. There is generally a well-ordered ranking of the stability of the different script integrity dimensions.

4.4 Case Studies

In this section, we provide examples of the types of changes in frequently-changing scripts that do not vary by user agent or time.

Data Changes. In the structural signature scheme, data changes cause signatures to change because the identity of an identifier includes its initialization value. This is because data changes can affect the execution and behavior of scripts, even if functions remain unchanged. Furthermore, they can be more complex than simple string changes; for example, variables can contain functions that will be dynamically loaded. A popular analytics script, Google Tag Manager (GTM), is one that often experiences changes in data values. It acts as a "loader" script that pulls in other scripts and provides data to them. We observe that `7-eleven.com.mx` uses GTM, and we witness changes every day in data variables that presumably live in the data layer (e.g., the variable `N.oh`). Others may act as configuration files; we find that `cnet.com` sources a script from the effective top-level domain (eTLD) `fastly.net` at a file path that ends in `/gpt_and_prebid/config.js`, and it primarily experiences changes in the property `settings['rules']`, which appears to define rules for ad prebidding.

Takeaways: The impact of data changes on the execution of a script varies, and there is no one-size-fits-all solution to handling data changes from the perspective of a client, without knowing the potential ramifications of the data change.

Code Transformations. Other scripts experience direct modifications to the code itself, instead of to only data literals. For example, such scripts might alternate the positions of function declarations or modify function calls. We observe an instance of the former on the domain `hiragana.jp`, which fetches a script from the origin `js.hs-scripts.com` at the path `/8947762.js`. From a comment that appears in some versions of the script, it self-identifies as a "HubSpot Script Loader." In addition to a changing data variable, which holds the URL for a script to be dynamically loaded, the position of the top-level function that contains the variable also changes, with no apparent pattern. This script was served from a content delivery network (CDN), but we did not observe a correlation between this characteristic and the transformations. Similarly, other scripts might regularly experience changes to the names of its functions. Such scripts include those found on the domains `timescolonist.com` (sourced from eTLD `facebook.com`), `jd.hk` (from `jd.com`) and `silive.com` (from `blueconic.net`). All three of these scripts are comprised of a single, global function call that is defined externally (by another script) — the function name varies, but the arguments remain the same.

Takeaways: Direct modifications to code present a different challenge than modifications to data values. A naïve solution to handle changes to data values can be to entirely exclude them from the integrity verification process, and script authors can attempt to verify data literal values used by their code with using their own knowledge. However, there is no straightforward, analogous naïve safeguard if the names of function calls are also excluded, particularly if there are scripts that access global objects exposed by other scripts.

5 (RQ2) PITFALLS OF STRICT INTEGRITY

In this section, we discuss crucial pitfalls in the usability of strict integrity mechanisms such as SRI and also include complementary analyses in Appendix B.

Correctness. We observe that 79% of all scripts never changed content. These scripts were provided by 76% of all script providers, and were found on 90% of the domains we were crawling. We emphasize that a visitor is protected from a third-party compromise if and only if *every script requested from that particular third party uses SRI.* Verifying the integrity of a *subset of scripts* from a third party offers partial protection at best, and is less secure than verifying the integrity of all scripts from that third party.

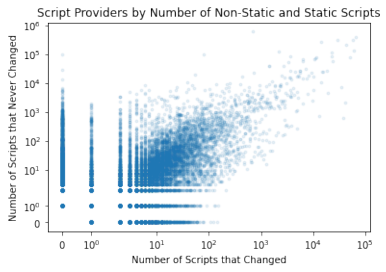


Figure 5: *Remote origins, by the static and non-static scripts they provide.*

In Figure 5, we visualize the ability to "apply SRI" to all 52,169 origins, by comparing the number of non-static-content scripts (x) with the number of static-content scripts (y). We note that there are primarily three classes of interest: (1) 44% of third parties serve only static scripts (points with $x = 0$), to which SRI is straightforwardly applicable; (2) 24% of third parties serve only changing scripts (points with $y = 0$), to which the usability of SRI ranges from somewhat usable to impractical depending on the variability of the scripts; and (3) 32% of third parties serve a mix of both, to which it is difficult to completely apply SRI. Using SRI for such third parties offers partial protection at best and no protection at worst. However, it may not be possible to apply SRI to all origins that provide only static scripts. If we revisit the 79% of all scripts that never changed content, we find that only 6.5% of them are delivered by providers that only serve static scripts, whereas 94% of them are delivered by providers that serve both static and dynamic scripts. On a randomly selected day in our time window in which 8,911 domains used SRI, we find that, for each of these domains, there were on average 0.98 third parties that provided scripts for which SRI was always used, 0.32 third parties that provided scripts for which SRI was used for only some scripts, and 15 third parties that provided scripts for which SRI was not used at all. In fact, 21% of these domains did not completely apply SRI to all the scripts from any one particular third-party provider.

On one hand, from the bimodal distribution of script content changes in Section 4.2, the 31% of all domains which requested for at least one script that changed every day cannot completely protect themselves against these providers. On the other hand, the 98% of all domains which requested at least one script that changed in content may find it difficult to apply SRI. We thus establish a range of the difficulty of applying SRI: at least 31%, and at most 98%, of the domains in our sample would find it difficult to protect themselves against third parties using strict integrity alone.

Takeaways: To correctly secure oneself against a remote third party requires verifying the integrity of *all scripts* received from them. A significant fraction (21%) of the already-small percentage of first parties that adopted SRI do not use it appropriately, and for every first party domain that used SRI, the number of third parties with no SRI protection is more than one order of magnitude larger than the number of third parties with partial or complete SRI protection. Furthermore, 24% of third party origins provide only non-static

Table 4: *SRI misconfigurations.*

Rank	Domain	Remote Script URL	Duration Observed	Description
1,136	cuny.edu	browsealoud.com/plus/scripts/ba.js	[2021-12-10, 2022-02-26]	*Asks remote party for SRI digest*
9,510	universalorlando.com	cdn.cookielaw.org/scripttemplates/otSDKStub.js	[2022-01-29, 2022-02-22]	Different digests for the same script
15,168	donorschoose.org	ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js	[2021-12-11, 2022-02-26]	Different digests for the same script
680,867	mcbw.de	mcbw.de/.../first-party-scripts.js	[2021-12-10, 2022-02-26]	Different digests for 14 first-party scripts
680,324	borlange.se	browsealoud.com/plus/scripts/ba.js	[2021-12-10, 2022-02-26]	*Asks remote party for SRI digest*
682,215	frisianflag.com	code.jquery.com/jquery-3.6.0.min.js	[2021-12-10, 2022-02-26]	Different digests for the same script

scripts, and 32% provide a mix of static and non-static scripts, making it even more difficult for a first party to secure itself against third party providers. Finally, at least 31%, and at most 98%, of the domains in our sample would find it difficult to straightforwardly protect themselves against third parties using strict integrity alone.

Misconfigurations. Due to the scale of our experiment, we were able to witness previously undocumented SRI misconfigurations on six domains. We found two major types: (1) obtaining the SRI digest of a remote script from its provider, and (2) using incorrect SRI digests for a script on some pages and correct digests on other pages — even for first-party scripts.

The first misconfiguration was found on a third-party script used by the domains `cuny.edu` (ranked 1,136) and `borlange.se` (ranked 680,324) while investigating how the digest for a script changed in the process of crawling both domains on 2022/02/10. We found that both domains include a `ba.js` script in the HTML of some pages that (1) first asks the URL `browsealoud.com/.../sri.json` for the SRI digest of a particular version of the `browsealoud.js` script, before (2) programmatically injecting the `<script>` element with the digest. This is an incorrect usage of SRI that does not provide any security guarantee, even if the initial `ba.js` script were to be requested using SRI (it did not use SRI), because a compromised provider would be able to change the script content and the digest accordingly.

The second misconfiguration is related to a shortcoming of the infrastructure used by domains to manage their dependencies. We observe that multiple pages of the same website would request the same script, but with different SRI digests. Fortunately, in the misconfigurations that we found, at least one of the digests was correct, but the other digests were either blank (i.e., the browser will not attempt to verify integrity) or wrong (i.e., the browser will not execute the script and report a failure in the developer console). These misconfigurations could continue for long periods of time; we witness short-lived cases that suddenly started and stopped after a few days, suggesting manual misconfiguration, but also cases that continued for months, suggesting the developers were not aware of the problem. Interestingly, we found 14 instances of this misconfiguration for first-party scripts on one domain, indicating that misconfigurations are possible, even with full control over the domain. Refer to Table 4 for more detail about these misconfiguration instances.

Takeaways: It is imperative for new standards, particular security standards, to possess built-in reporting mechanisms. Surprisingly, it was possible for SRI misconfigurations of first-party scripts to remain unnoticed for multiple months.

6 RELATED WORK

To our knowledge, there exists no other study that presents as comprehensive a report of script changes in the modern web through the lens of integrity. Our work focuses on enumerating the severity of the integrity problem, as compared to prior works that proposed defenses to address it. In this section, we review such integrity mechanism proposals, as well as existing measurement studies that contribute crucial findings in this space.

Our work primarily relates to studies of SRI usage. Kumar et al. [13] found that less than 1% of sites use SRI in their enumeration of security challenges in a growing web, and Shah and Patil [22] found that less than 1% of sites enforce SRI on all subresources in their measurement of its adoption in 2018, two years after its release. Chapuis et al. followed in this line of work with a thorough measurement of the adoption of SRI using a much larger set of web pages, over the span of multiple years, and also surveyed developers to understand how SRI is perceived and used, finding that the driving factor behind its growth seems to be copy-paste behavior from developers [5]. Similarly, Steffens et al. conducted a targeted study [27] to understand how third-party script inclusions are impairing the use of Content Security Policies (CSP) [26] and SRI. In particular, they find that high-profile parties often randomize parts of their scripts, hindering the ability to use SRI. We argue that these studies measure the adoption and usage of an implementation of an integrity mechanism, instead of the *usability of integrity verification.* In contrast to the statistics reported by Soni et al. that frequently-changing scripts only accounted for 0.62% of all scripts found on 69 domains [24], we find that frequently-changing scripts are no longer a rare exception: 16% of scripts, located on 31% of all domains, change content every day, and that SRI would not provide satisfactory security guarantees even if every static script were protected with SRI, because providers serve a mix of static and non-static scripts.

There also exists a line of work that studies dependencies on the Internet. Dell'Amico et al. [7] performed a large-scale study of both active and passive DNS data to discover the dependency relationships between websites and Internet services, finding that the service ecosystem is dominated by a handful of popular providers. Similarly, Simeonovski et al. [23] evaluated the impact of multiple attacks, including the distribution of malicious JavaScript content, using their technique to model services, providers, and dependencies as a property graph. Nikiforakis et al. [20] performed one of the first large-scale studies of JavaScript in particular, and a recent study from Mitropoulos et al. analyzed the evolution of JavaScript code in the wild [16]. Overall, these studies seek to characterize the relationships between websites and their dependencies, whereas our work focuses on breaking down how a particular type of dependency changes. With respect to the finding that scripts have frequent update cycles reported by Mitropoulos et al., it corroborates our own, but this finding is in the context of their study of the evolution of software quality issues, as opposed to how such frequently-changing scripts change and present a challenge to integrity verification.

In addition, studies regarding JavaScript integrity verification are also closely related. We broadly group integrity verification mechanisms as *strict* or *relaxed*, depending on the rigidity of their content matching. Strict integrity verification mechanisms include the SRI standard and more recent proposals from Nakhaei et al. [19] and Mignerey et al. [15]. Nakhaei et al. proposed a protocol that would require clients to treat received script content as raw text, extract the first line containing a hash of the rest of the text (i.e., the actual script content), and verify the hash before executing the script, which does not address our threat model. Mignerey et al. proposed a new web ecosystem component with trusted third parties to produce and store files that contain the hashes of scripts that are allowed to run on a page. Relaxed integrity mechanisms include structural signatures from Soni et al. [24] and contextual script fingerprints from Mitropoulos et al [17] as previously mentioned. We distinguish this work as follows: (1) we focus on enumerating the severity of the problem they intend to address, and (2) we shed light on potential reasons as to why such defense mechanisms are not applicable in the modern web. We demonstrated the importance of accounting for frequently-changing scripts because of their wide use and presented results on the (in)stability of various script characteristics, including contextual attributes and structural signatures, that would contribute to the overall instability of their proposals.

7 (RQ3) DISCUSSION

Strict integrity alone, such as SRI, cannot account for frequently-changing scripts (including those that are dynamically generated) and scripts with dynamic URLs. We re-emphasize that *frequently-changing scripts appear to change in consistent manners*, holding true even for complex, ad-related scripts that appear on filter lists such as EasyList and EasyPrivacy, and there is generally a well-ordered ranking of the stability of different script (request) dimensions. However, even if two scripts change in the same dimension, e.g., a data string change, this does not guarantee the same change in execution. As such, it is infeasible for a user agent to distinguish "subtle," benign changes from malicious ones *without additional knowledge*. This motivates the need for a new protocol to provide clients the context that is necessary to assess the potential ramifications of script changes.

A more immediate stopgap is to ensure the integrity of higher-order elements (to address the challenge posed by scripts with dynamic URLs that may also change in content) or by positioning the integrity mechanism such that it does not rigidly rely on the location of a script. The former is similar to the approach used by the `strict-dynamic` keyword in the Content-Security Policy (CSP) standard [31], which establishes an explicit trust in an element and then transitively trusts those that are trusted by the initial element. The latter can be done by associating integrity artifacts with third-party origins, as opposed to individual script URLs (e.g., defining the set of allowed script signatures for a particular third party).

7.1 Looking Forward

Our work demonstrates that the problem of verifying the integrity of modern JavaScript is complicated by a number of factors that each exacerbate the overall issue, and that the current standard, SRI, is lacking critical qualities. Scripts will continue to change frequently, and do so in a variety of content-related, contextual, and seemingly random manners. If the modern web continues to lack a robust and practical integrity verification mechanism, domain administrators will continue to run the risk of having their users' security and privacy compromised by a supply chain attack via malicious JavaScript.

Orthogonal to the design of a more robust integrity mechanism, we argue that a simple reporting feature in SRI, such as the `report-to` directive in CSP [31], can enable administrators to gain visibility into potential supply chain attacks and broken functionality, thus preventing long-term SRI failures and misconfigurations.

7.2 Limitations

We acknowledge that our methodology has inherent limitations, but we are confident in the high-level, stable findings that we report. The most prominent limitation is that our findings are necessarily constrained by the scripts that are collected. A more intricate approach to mask the signals that are unique to headless browsers could have resulted in the collection of more scripts, and the choice of a different headless browser (e.g., Firefox) might have yielded the collection of different versions of scripts. In addition, we may have missed additional findings that can only be observed over a much longer period of time (i.e., years), or in particular classes of scripts. We attempted to mitigate these potential issues by, in part, mimicking full browsers and performing the analysis over a period of a few months.

Apart from this, we focused on "external" scripts from a third-party origin, but it may be the case that two different origins are not *entirely* external from one another (e.g., two subsidiaries of the same entity). Although we did not account for this (using such concepts as the extended same party [27]), we argue that the extent of the isolation of such domains is entirely dependent on the configuration.

8 CONCLUSION

In this paper, we demonstrated the difficulty of properly ensuring the integrity of scripts in the modern web with a large-scale study of the script landscape. We conducted our experiments using a scalable and efficient architecture that enables a longitudinal study of the manners in which scripts change, at the scale of the web. We empirically demonstrated that frequently-changing scripts should be considered first-class citizens of the web ecosystem, and that the ways in which scripts change tend to remain constant over time. Furthermore, we performed an analysis of the usability and applicability of strict integrity at the granularity of the script providers themselves, and discovered that it cannot achieve widespread use nor provide satisfactory guarantees. Finally, we conclude that it is infeasible for a client to distinguish benign changes from malicious ones without external knowledge, motivating the need for solutions where clients are provided with the necessary context to assess the potential ramifications of script changes.

ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research (ONR) under grant N00014-20-1-2720 and by the National Science Foundation (NSF) under grants CNS-1941617, CNS-2211575, and CCF-2153297.

REFERENCES

[1] John Althouse, Jeff Atkinson, and Josh Atkins. 2017. JA3 - A method for profiling SSL/TLS Clients. https://github.com/salesforce/ja3.
[2] berstend. 2018. Puppeteer Stealth Plugin. https://github.com/berstend/puppeteer-extra.

[3] VMware Carbon Black. 2019. *Global Incident Response Threat Report-The Ominous Rise of "Island Hopping" & Counter Incident Response Continues.* Technical Report.
[4] Bootstrap. 2013. Bootstrap. https://getbootstrap.com/.
[5] Bertil Chapuis, Olamide Omolola, Mauro Cherubini, Mathias Humbert, and Kévin Huguenin. 2020. An empirical study of the use of integrity verification mechanisms for web subresources. In *Proceedings of The Web Conference 2020.* 34–45.
[6] Quan Chen, Peter Snyder, Ben Livshits, and Alexandros Kapravelos. 2021. Detecting filter list evasion with event-loop-turn granularity javascript signatures. In *2021 IEEE Symposium on Security and Privacy (SP).* IEEE, 1715–1729.
[7] Matteo Dell'Amico, Leyla Bilge, Ashwin Kayyoor, Petros Efstathopoulos, and Pierre-Antoine Vervier. 2017. Lean on me: Mining internet service dependencies from large-scale dns data. In *Proceedings of the 33rd Annual Computer Security Applications Conference.* 449–460.
[8] Nurullah Demir, Matteo Große-Kampmann, Tobias Urban, Christian Wresnegger, Thorsten Holz, and Norbert Pohlmann. 2022. Reproducibility and Replicability of Web Measurement Studies. *TheWebConf 2022* (2022).
[9] Google. 2008. Chrome DevTools Protocol. https://chromedevtools.github.io/devtools-protocol/.
[10] Google. 2014. Google Tag Manager. https://tagmanager.google.com/.
[11] Owen Hughes. 2021. Top programming languages: Most popular and fastest growing choices for developers. https://www.zdnet.com/article/top-programming-languages-most-popular-and-fastest-growing-choices-for-developers/.
[12] Shahzad Khan. 2021. What makes JavaScript so popular. https://generalassemb.ly/blog/what-makes-javascript-so-popular/
[13] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J Alex Halderman, and Michael Bailey. 2017. Security challenges in an increasingly tangled web. In *Proceedings of the 26th International Conference on World Wide Web.* 677–684.
[14] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019).* https://doi.org/10.14722/ndss.2019.23386
[15] Josselin Mignerey, Cyrille Mucchietto, and Jean-Baptiste Orfila. 2020. Ensuring the Integrity of Outsourced Web Scripts. In *ICETE (2).* 155–166.
[16] Dimitris Mitropoulos, Panos Louridas, Vitalis Salis, and Diomidis Spinellis. 2019. Time present and time past: analyzing the evolution of JavaScript code in the wild. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR).* IEEE, 126–137.
[17] Dimitris Mitropoulos, Konstantinos Stroggylos, Diomidis Spinellis, and Angelos D Keromytis. 2016. How to train your browser: Preventing XSS attacks using contextual script fingerprints. *ACM Transactions on Privacy and Security (TOPS)* 19, 1 (2016), 1–31.
[18] Richard Moore. 2016. Ethers.js. https://ethers.org/.
[19] Kousha Nakhaei, Fateme Ansari, and Ebrahim Ansari. 2020. JSSignature: eliminating third-party-hosted JavaScript infection threats using digital signatures. *SN Applied Sciences* 2, 1 (2020), 1–11.
[20] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security.* 736–747.
[21] Elastic NV. 2010. Elasticsearch. https://www.elastic.co/.
[22] Ronak Shah and Kailas Patil. 2018. A measurement study of the subresource integrity mechanism on real-world applications. *International Journal of Security and Networks* 13, 2 (2018), 129–138.
[23] Milivoj Simeonovski, Giancarlo Pellegrino, Christian Rossow, and Michael Backes. 2017. Who controls the internet? analyzing global threats using property graph traversals. In *Proceedings of the 26th International Conference on World Wide Web.* 647–656.
[24] Pratik Soni, Enrico Budianto, and Prateek Saxena. 2015. The SICILIAN defense: Signature-based whitelisting of web JavaScript. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security.* 1542–1557.
[25] Sourcedefense. [n.d.]. Recent 3rd party JavaScript attacks. https://sourcedefense.com/resources/blog/recent-3rd-party-javascript-attacks/.
[26] Sid Stamm, Brandon Sterne, and Gervase Markham. 2010. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World wide web.* 921–930.
[27] Marius Steffens, Marius Musch, Martin Johns, and Ben Stock. 2021. Who's Hosting the Block Party? Studying Third-Party Blockage of CSP and SRI. In *Network and Distributed Systems Security (NDSS) Symposium 2021.*
[28] Tobias Urban, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. 2020. Beyond the front page: Measuring third party dynamics in the field. In *Proceedings of The Web Conference 2020.* 1275–1286.
[29] Lionel Sujay Vailshery. 2022. Most used languages among software developers globally 2021. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/
[30] W3C. 2016. Subresource Integrity. https://www.w3.org/TR/SRI/.
[31] W3C. 2021. Content Security Policy Level 3. https://www.w3.org/TR/CSP3/.

Appendices

A (RQ1) CONSISTENT PATTERNS OF CHANGE

In this section, we include additional analyses to shed light on the consistent patterns of change from a high-level perspective of changes in all scripts, and in scripts with dynamic URLs.

High-Level Perspective of Changes. We present a high-level overview of all daily script changes throughout 77 days in Figure 6. We consider four natural types of "changes" in scripts: scripts can stay the same (in content), change (in content), disappear (i.e., not encountered), or (re)appear. The first two are changes in raw content, whereas the last two are changes in location, but they can also be changes in content. Changes are computed on a day-to-day basis, taking the set of script contents received while browsing a particular domain D on day d and comparing it with the contents received while browsing D on day $d-1$. The x-axes in both plots represent the relative time difference, in days, from the first day we observed a script. Scripts are identified in a slightly different manner in both plots: the upper plot identifies scripts in the manner described earlier (by the script URL and the `Referer`), whereas the lower plot uses the origin of the script URL instead of the script URL, in addition to

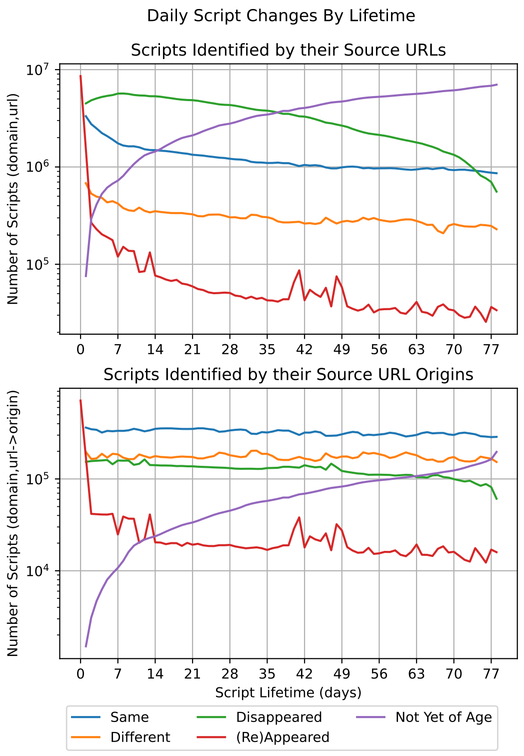


Figure 6: *Daily changes in scripts found while crawling our sample of domains. We experienced some instability in our crawling infrastructure on day 38, causing an artificially large spike in our data — the data points for this and the following date have been smoothed to account for this.*

the `Referer`. In short, the upper plot considers changes in content received from script URLs, whereas the lower plot considers changes in content from script provider origins.

Takeaways: From Figure 6, we can infer that:

- dynamic script URLs in the same origin contribute to the difference in the behaviors of the *Disappeared* or *(Re)appeared* buckets in the two plots.
- there are long-lived script URLs (and their origins), because the four types of changes generally stabilize over time.
- there are "new" script identifiers that appear every day, even though we are crawling the same set of pages, on the same set of domains, because the number of scripts in the *Not Yet of Age* bucket increases with time.
- URL identifiers are not as stable as URL origin identifiers, because there are likely many short-lived URL identifiers that contribute to a significant portion of the *Disappeared* bucket in the upper plot.

Content Changes in Dynamic-URL Scripts. We seek to investigate whether dynamic-URL scripts are not only changing their locations, but also changing their content. In particular, we consider dynamic-URL scripts from the top 10 domains that contributed the most of them. However, because we do not have a reliable method of identifying such scripts, we are unable to compare differences in content between two individual scripts. Instead, we group these dynamic URL scripts by the `Referer`, or the domain that we were crawling when we observed these script requests. For each day, we extract the structural signatures from all dynamic-URL scripts for each domain, and we compare the set of signatures with that of the prior day to obtain a metric that describes the change in the two sets of signatures. If a structural signature is present one day but not present the prior day, it contributes +1; if a structural signature is not present one day but present the prior day, it contributes −1. We then plot the distribution of this metric for all domains over time and show our results in Figure 7. We observe that different domains exhibit different behavior: some domains do not show much change in the set of script signatures every day, whereas others might experience up to a few hundred changes on a given day. Overall, scripts that have dynamic URLs can also have dynamic content, which is a nontrivial complication to integrity verification. We note that regardless of the choice of using the original structural signature or the augmented, no-literal version that we used in Section 4, the resulting plots are nearly identical.

Takeaways: Scripts with dynamic URLs pose a distinct challenge from the one presented by frequently-changing scripts with static URLs, because dynamic-URL scripts can also change in content.

B (RQ2) PITFALLS OF STRICT INTEGRITY

In this section, we include additional analyses to shed light on the pitfalls of strict integrity through analyzing the popularity of frequently-changing scripts and the rankings of their providers.

Popularity of Frequently-Changing Scripts. If most scripts used by websites are static, strict integrity could provide security guarantees in most cases. However, we find that a significant portion of scripts changes frequently, regardless of the URL lifespan (an upper bound on the number of changes). In Figure 8, a CDF for each lifespan class is produced with (x,y) values indicating that the URLs of scripts that changed content more than x times comprise y of all the unique URLs of all scripts with the same lifespan. For scripts with static URLs, 68% of their script URLs change their content at least once within seven days (12 or more content changes); for scripts with a URL lifespan of 60 days, it is 33% (nine or more content changes); for scripts of 40 days, it is 41% (six or more content changes); and for scripts of 20 days, it is 57% (3 or more content changes).

Takeaways: Regardless of the duration for which a script URL appears on a website, it is much more likely for its content to change, as opposed to remain static, throughout its appearance. We found that 68% of the URLs of daily-observed scripts change content at least once a week.

Rankings of Frequently-Changing Script Providers. It may be concerning if the provider of a frequently-changing script has a low popularity ranking. Although domain ranking may not always be an accurate proxy for its security posture, we can estimate that domains that are very high ranked have a better posture than those that are very low ranked [20]. We observe this split in the rankings for the 1,379 unique eTLDs of script providers for the 38,030 scripts that change daily in Figure 9. Providers that were not ranked within the Tranco top 1 million were assigned the same rank above 1 million and appear in the same bin in the histogram. We note that the significant

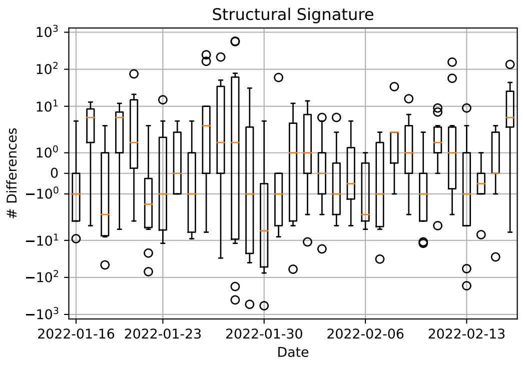


Figure 7: *Distribution of the number of changes in the set of structural signatures in the sample of scripts with dynamic URLs every day. Dynamic URL scripts also experience content changes.*

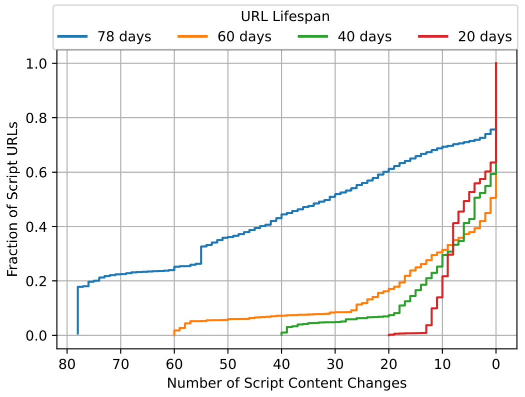


Figure 8: *Reverse CDF of scripts by their number of content changes, grouped by URL lifespan because it is an upper bound on the number of changes. Scripts that change the most frequently comprise a large portion of the unique script URLs, regardless of lifespan.*

portion of providers that are low-ranked — 809 (59%) low-ranked providers deliver 1,979 (5.2%) daily-changing scripts — creates additional cause for concern, because it becomes even more imperative to have an integrity verification mechanism that can account for this class of scripts.

Takeaways: If frequently-changing scripts cannot be handled by integrity verification mechanisms, their providers are more likely to be targeted in supply-chain attacks. The majority of such providers with low rankings (59%) is concerning because this may indicate they have weaker security postures.

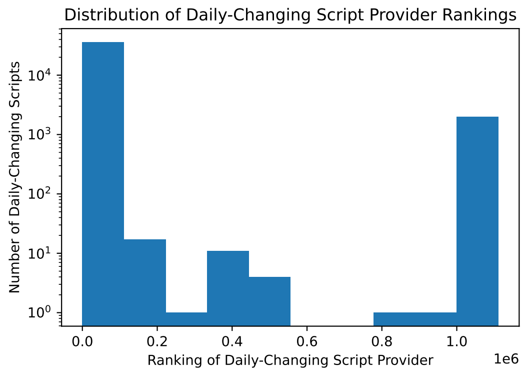


Figure 9: *Distribution of the scripts that change every day and do not vary by user agent or time, by the ranking of the eTLD of the provider. Many (59%) providers are low ranked.*