Who's Hosting the Block Party? Studying Third-Party Blockage of CSP and SRI

Marius Steffens*, Marius Musch†, Martin Johns†, and Ben Stock*
*CISPA Helmholtz Center for Information Security: {marius.steffens,stock}@cispa.de
†TU Braunschweig: {m.musch,m.johns}@tu-braunschweig.de

***Abstract*—The Web has grown into the most widely used application platform for our daily lives. First-party Web applications thrive due to many different third parties they rely on to provide auxiliary functionality, like maps or ads, to their sites. In this paper, we set out to understand to what extent this outsourcing has adverse effects on two key security mechanisms, namely Content Security Policy (CSP; to mitigate XSS) and Subresource Integrity (SRI; to mitigate third-party compromises) by conducting a longitudinal study over 12 weeks on 10,000 top sites. Under the assumption that a first party wants to deploy CSP and SRI and is able to make their code base compliant with these mechanisms, we assess *how many sites* could fully deploy the mechanisms without cooperation from their third parties. For those unable to do so without cooperation, we also measure *how many third parties* would jointly have to make their code compliant to enable first-party usage of CSP and SRI.**

To more accurately depict trust relations, we rely on holistic views into inclusion chains within *all* pages of the investigated sites. In addition, based on a combination of heuristics and manual validation, we identify different eTLD+1s belonging to the same business entity, allowing us to more accurately discerning parties from each other. Doing so, we show that the vast majority of sites includes third-party code which necessitates the use of `unsafe-inline` (75%) or `unsafe-eval` (61%), or makes deployment of `strict-dynamic` impossible (76%) without breakage of functionality. For SRI, based on the analysis of a single snapshot (within less than 12 hours), we also show that more than half of all sites cannot fully rely on SRI to protect them from third-party compromise due to randomized third-party content.

I. Introduction

The Web, as we know it today, is deeply intertwined with our everyday life. It allows us to perform essential tasks, such as keeping in touch with our beloved ones through social media, and even serves us feature-rich business appliances. However, since the advent of the *Web 2.0*, there was a rapid growth in complexity on the client side [33]. Notably, the amount of different contributors from which scripting resources are included follows this increase, resulting in a continually rising reliance on third parties. For example, the Web site of a coffee shop can refer to an external library to incorporate an interactive map showing the quickest route to the next store. The coffee shop can thus rely on a third party to take care of this specific part, enabling the first party to focus on the needs of their core functionality. Other use cases include advertisement, analytics, or simply hosting a widely used library in a single place, reducing traffic on one's site.

The increasing reliance on third parties to provide functionality to first-party sites naturally comes with risks. In particular, including scripts from others allows such third parties to add whatever code they deem necessary and even delegate this privilege to arbitrary additional external parties. Numerous research works in this area have shown that this reliance on third parties increases the attack surface of first-party sites, e.g., via third-party compromise due to bad security practices [22], the addition of malicious code via intertwined inclusion chains [2, 8], and the introduction of client-side XSS [18, 31, 32].

Hence, it is in a site developer's best interest to defend their site against such threats. Specifically, we focus on two key mechanisms to protect Web sites: Content Security Policy (CSP) and Subresource Integrity (SRI). CSP [29] primarily aims to mitigate the impact of XSS vulnerabilities. In contrast, SRI [20] aims to secure including sites against compromise of third-party servers by only executing scripts that match the cryptographic hash attached to their definition. Unfortunately, both mechanisms lack widespread deployment [1]. Up to 95% of deployed CSPs are utterly insecure [3, 39], and rolling out CSP was recently shown to be a lengthy process in which some sites ultimately give up and fail to arrive at a meaningful policy [27]. New CSP features, such as `strict-dynamic`, are intended to make a secure deployment of CSP easier for the first party. Specifically, the authors envision *"with such a policy, the owner would need to add nonces to static <script> elements, but would be assured that only these trusted scripts and their descendants would execute. This mode of deploying CSP can significantly improve the security of a policy and facilitate adoption"* [39]. Nevertheless, even though the feature was added in 2016 to the CSP standard, it has not been adopted by many sites, and script-controlling policies are still as insecure as ever [27]. Similarly, while SRI's popularity is slowly increasing, this is primarily due to widespread libraries like jQuery [4], leaving many other resources susceptible to cause severe harm in case of compromised third parties.

The main focus of our work is to assess *how many sites* are blocked from meaningful usage of CSP and SRI through *how many third parties*; under the assumption that first parties

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021, San Diego, CA, USA
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24028
www.ndss-symposium.org

want to secure their site (and are able to make their own code compatible, especially with CSP). Virtually all prior research on third parties [2, 8, 10, 22, 34] has used the notion of an eTLD+1 to reason about parties. This assumption, however, does not hold true as it is common for modern sites to split their application logically across multiple eTLD+1s. Instead, we define parties by the entity that operates them, e.g., Google for Youtube and Doubleclick. We derive two heuristics to identify candidates for our definition of an *extended Same Party* and manually vet them to avoid false positives.

With this improved notion of a party, we tackle our main research question through an extensive 12-week experiment of the Tranco top 10,000, in which we collect inclusions, attribute them to the respective parties, and observe the behavior the different parties exhibit. In doing so, and assuming a first party that is willing to tackle the necessary modifications to their own codebase to enable a meaningful CSP, we show that the instability in inclusions through third parties limits the applicability of host-based allowlists, the only universally supported mode of CSP (as even modern browsers like Safari lack `strict-dynamic` support). Similarly, the usage of APIs like `eval` and the parser-inserting addition of script elements and event handlers hinders deployment of policies without the `unsafe-eval` and `unsafe-inline` keywords, and hampers roll-out of nonce-based `strict-dynamic` policies. We also highlight that high-profile third-party inclusions such as Facebook or Doubleclick often randomize minuscule parts of the script content, rendering the hash-based SRI deployment impossible. With both these aspects, even a party willing to make their own code base compliant with these security mechanisms, will not be able to have meaningful security without suffering from breakage of third-party functionality.

In sum, our paper makes the following contributions:

- Based on observed inclusion relations, we derive a new notion of *party* beyond eTLD+1, dubbed *extended Same Party* in Section IV. We then show that our new notion is much closer to reality than using eTLD+1s, highlighting the need for such an improved notion for the modern Web (Section V).
- We measure how the behavior of first, third, and delegated code impacts the deployment of CSP in Section VI. In particular, we study how the fluctuation in included hosts renders host-based CSPs infeasible, how often sites would need to use the trivially insecure `unsafe-inline` keyword, and how sites cannot use `strict-dynamic` without causing breakage in their third-party dependancies.
- In Section VII, we analyze the feasibility of restricting the behavior of a site's codebase to the script hash using SRI and show that the fluctuation of script content hosted at static URLs does not allow the majority of sites to pin their third-party dependencies. Furthermore, we show that in the wild SRI's protection is frequently undermined by unpinned inclusions performed by pinned resources.
- Based on our empirical analyses, we provide calls to action to both third-party script providers and browser vendors. Furthermore, to aid developers in assessing the adverse effects of third-party scripts, we open-source SMURF, which allows to attribute CSP-incompatible behavior to hosts.

II. Technical Background

In this paper, we refer to a registerable domain (e.g., `bbc.co.uk` or `google.com`) as an *eTLD+1* (effective top-level domain+1). We use the term interchangeably with *site*, aligned with, e.g., the notion of same-site that browsers use for security mechanisms like site isolation [25]. In addition, we use the term *disconnect* to reason about how connected a first party is to third-party code; i.e., whether they directly include such code or whether any of the included parties added the additional scripts.

A. JavaScript Inclusions

The Web's core security concept is the Same-Origin Policy, which ensures that a script can only access resources from the same origin (protocol, host, and port). JavaScript inclusions are partially exempt from this rule, as any HTML document may include scripts from other origins. While the content of the scripts cannot be read, the JavaScript engine will execute these scripts; importantly, in the origin of the *including* document. JavaScript can, in turn, add additional scripting resources, be it through writing script tags or event handlers through `document.write`, invoking `eval` to convert a string to code, or programmatically adding scripts to the DOM through `document.createElement` and `appendChild`. By default, inclusions cannot be restricted, i.e., any included script can add additional content to its liking.

B. Content Security Policy

In its original form, the Content Security Policy (*CSP*) was meant to mitigate the effects of Cross-Site Scripting and enable a developer to limit the resources which could be loaded into their site [29]. This is achieved by providing an allowlist of origins from which external content can be included, in combination with disallowing potentially dangerous constructs such as inline scripts, event handlers, and `eval` by default. To allow for backward compatibility in using those unsafe practices while rolling out a strict CSP, `unsafe-inline` and `unsafe-eval` are part of the CSP specification. Deploying a policy with `unsafe-inline` essentially allows an attacker abusing an injection vulnerability to insert their script content directly as an inline script; such policies cannot mitigate XSS. At the same time, as removing any inline script seemed infeasible, CSP added nonces and hashes [36]. Through this, developers can attach a random nonce to each script (as contained in the CSP), making them executable; similarly, scripts can be allowed through their hash sum. It must be noted, though, event handlers cannot be allowed in this fashion. The only option to achieve is to add the `unsafe-hashes` [37] attribute to the policy, which enables event handlers to be executed if their hash is explicitly allowed.

Orthogonally, if a site operator needs to use `eval`, they have to resort to `unsafe-eval`.

Weichselbaum et al. [39] proposed `strict-dynamic` to alleviate the burden of keeping a CSP up to date with all the hosts being added by third parties. If this mode is enabled, any script that is allowed through a hash or a nonce can *programmatically* add additional scripts, i.e., by using `createElement` and `appendChild`, but not `document.write`. Notably, when this option is enabled, any host-based allowlist is disabled, meaning that even inclusions from the same host must be done programmatically.

C. Subresource Integrity

While CSP aims to mitigate script injection attacks, Subresource Integrity (*SRI*) was designed to disarm malicious modification of externally included resources, e.g., through compromised Content Delivery Networks or via in-transit alteration of script code done by network-based attacks [20]. To take advantage of this feature, the including party has to specify the hash of the expected content in a script's `integrity` attribute. In case of a mismatch between the hash of the content and the integrity value, the browser refuses to execute the script. Given the nature of the mechanism, a single-byte change in the included script will lead to mismatching hashes, effectively disabling the modified script. Hence, for a script to be SRI-compatible, it must not change; a prime example of an SRI-pinnable script is jQuery, which can be included centrally from jQuery.org through an explicit version number in the URL, which never changes its content. Recently, a work of Cherubini et al. [5] proposed to extend the capabilities of SRI to also allow developers to pin the contents of downloads which they link to on their own page, reducing risk of erroneous checksum verification of end-users.

III. Experiment Parameters

To answer our main research question, namely whether first parties can simply change *their own* codebase to allow for seamless integration of CSP and SRI, we utilize the Tranco [12] list from January 13, 2020, to extract the 10,000 highest ranking sites[1]. To mitigate the impact of a single party with multiple internationalized domains (e.g., Google with around 80 different TLDs) on our results, we used Tranco's feature to group together those sites belonging to one organization.

We set out to analyze not only a single snapshot of the Web's tangled nature, but instead also to investigate the rate of change observable throughout a prolonged period. Therefore, we ran crawls once a week from January 13th through March 30th, 2020. For each crawl, our crawlers visited the start pages from the fixed list and followed every *same-site* link. To avoid influences of stale URLs, we repeat this process every time, limiting ourselves to a maximum of 1,000 pages per site. On average, each crawl yielded around 1 million URLs. For results that do not consider the longitudinal aspect of our data collection, we report on the data gathered in our first crawl. Overall, we could find that of the 10,000 Tranco entries, we could only analyze 8,389 by connecting to the website by following the link *http://entry*. In 493 cases, we hit a timeout in our crawling infrastructure. Besides sites that take too long to visit, we could find hints that some sites behaved differently when crawling them from our analysis machines compared to our home network. We expect that our public IP addresses used are known to host crawlers, and we do not take specific measures to conceal our traffic as human-generated. We were unable to connect to 603 entries because of network-level issues, such as NXDOMAIN, connection refusals, or certificate errors. Another 515 sites redirected our crawler to another site, which we therefore also excluded from our analysis. We can find 348 sites that do not include any scripting resources at all. Manual investigation showed the lack of scripting resources was mostly due to blank pages (again likely as our IP is known as a crawler). Notably, we also found instances in which Web sites refused us to access the real content, e.g., at `https://www.radio.com`, which instead showed a static warning page indicating unavailability for our geolocation. Ultimately, this leaves us with **8,041** sites with any script resource, which we consider throughout our analyses.

[1] https://tranco-list.eu/list/3Q2L

IV. An Improved Notion of Disconnect

In order to understand which parties are introducing code that is incompatible with CSP and SRI, we first need to discuss how we can reliably attribute code locality to the varying parties that jointly contribute to the overall code base of sites. We first highlight technical challenges that need to be considered for collecting precise *inclusion relations* (i.e., which party includes scripts from which other parties) on the level of single scripts. We then leverage the resulting inclusion trees to analyze co-occurrence patterns, which serve as heuristics for candidate hostname pairs that are likely to be affiliated. Lastly, we discuss how we can use these patterns in uncovering which hostnames actually belong to the same entity using manual efforts for the final verification and how we can quantify disconnect of code based on a holistic view into a given site.

A. Collecting Inclusion Relations

The first step in answering our research questions is the collection of real-world inclusions. While previous work has already developed the first tools to capture inclusions [11], we build upon their capabilities to address previously unaddressed issues related to state-of-the-art JavaScript features. These include asynchronous execution as well as incorrect attribution of inclusions to well-known libraries. In the following, we present how we address the shortcomings of prior works. In the spirit of open science, our implementation will be open-sourced as part of a lightweight analysis technique which we coin SMURF (see Section VIII-C), helping developers to uncover problematic inclusion decisions and allowing other researchers to compare against our work.

1) Precisely Capturing Inclusion Relations: To capture inclusion relations, we reconstruct which entity initiated a particular script inclusion using call stack traces. Although this has been done by prior work [11], through modern JavaScript features such as Promises, call stacks can be *asynchronous*. Based on the available implementations, this has not been considered before. To address this shortcoming, rather than relying on the stack trace available through regular JavaScript, we resolve all preceding stack traces via the DevTools' `Debugger.getStackTrace` API with the current trace's `parentId` in the arguments to get the *full* chain of events that led to a given inclusion. While this change may appear minuscule, given the increasing language support, e.g., async/await in ECMAScript 2017 [7], we believe this to be an important point to consider for our as well as all future work.

2) Correctly Handling Libraries: Usually, one would tie the notion of an inclusion's initiator to the top-most entry of the call stack. However, modern libraries present throughout the top sites provide asynchronous execution functionality, e.g., jQuery's `$(document).ready(callback)`. When called, jQuery stores the function pointer to `callback` and retrieves and subsequently executes the function when the document has finished loading. This delayed execution leads to the top of the stack being jQuery; hence, any inclusion conducted by the callback function would incorrectly be attributed to jQuery. In fact, artifacts[2] published by Lauinger et al. [11] even highlight cases in which the included jQuery script seemingly includes further inline scripts. Manual analysis of all the libraries (as classified by `retire.js` [23]) shows that no library by itself conducts further inclusions. Given this observation, we assume the first non-library script contained in an execution trace to be the actual initiator. Using this notion allows us to accurately infer the culprits behind actions, i.e., inclusion relations and API usage, even in the case of omnipresent libraries acting as confused deputies.

B. The Extended Same Party (eSP)

With this precise inclusion information, we can now turn towards understanding which hostnames actually belong to the same entity. This is necessary for two aspects of our analyses, namely to differentiate between first and third party (to count how many sites are affected by third parties) as well as to differentiate between different third parties (to count how many third parties affect a given site). In addition, this enables us to reason about delegation of trust, i.e., when a third party includes scripting content from another third party, which is important to understand whether a direct business relationship exists between a first and a third party. Related research [10] used the notion of an eTLD+1 to differentiate different parties; however, modern practices of first-party CDN's (e.g., `facebook.com` and `fbcdn.net`) or the logical separation of content (e.g., `doubleclick.net` and `googleadservices.com`) highlight the need for a refined notion that does not rely on domain labels alone.

[2]The jQuery in the lower-left corner at https://seclab.ccs.neu.edu/static/projects/javascript-libraries/causality-trees/modernfarmer.com/

Naturally, there is no ground-truth list of all domains belonging to a particular entity. Still, there exists a curated list of domains belonging to the same entity [13, 14] which is used as part of a tool named webXray [15]. Unfortunately, we could see that those lists frequently miss connections among two hostnames, e.g., `twitch.tv` and `twitchcdn.net`, which is to be expected as those lists are not explicitly crafted for our dataset. Therefore, we need to mine our dataset for more of such connections to attribute hostnames to entities accurately.

While clustering approaches based on TLS certificates or IP ranges appear meaningful to achieve such a mapping, we experimentally determined that such approaches yield high numbers of both false positive and negatives, e.g., through shared hosting (through Cloudflare) as well as disjunct IP ranges for different domains of the same entity (such `newrelic.com` and their CDN `nr-data.net`). We instead apply a semi-automatic approach, which involves relying on the observed inclusion relations in the wild and is complemented by a researcher validating all results manually. This way, our approach does not yield false positives (in the sense of two eTLD+1s flagged as belonging to the same party when, in fact, they are not). Naturally, any such empirical analysis yields imprecisions. However, as we show in Section V, the notion provides a much better upper bound for the number of third parties included in Web sites compared to relying on eTLD+1s.

As the first step, in uncovering further same-party domains, we look for eTLD+1s that are commonly used together in inclusions, such as `doubleclick.net` and `googleadservices.com`. Based on the crawl data from all our crawls (see Section III), we find combinations of two eTLD+1s with an inclusion relation on at least 10 sites. Based on this list of 908 combinations, we manually investigate their relation. In several cases, this is trivial, such as the example mentioned above. In other cases, this requires additional checks, such as for IP ranges of the involved domains, up to the manual inspection of the sites themselves (e.g., their imprints). This enables us to find pairs like `cookielaw.org` and `onetrust.com`, which are operated by the same entity/party.

While the previously outlined approach allows us to find large CDN providers, it does not yet allow us to find individual sites that have their own CDN. To find these, we analyze our collected inclusions to see cases in which a first party (identified by its eTLD+1) directly included content from a different eTLD+1 (the potential CDN). For each potential CDN, we check if it is also used on any other site we analyzed and only consider those domains which are exclusively used by one site. Furthermore, observations of the collected data indicated that keywords such as `img`, `cdn`, or `static` were often part of CDN domain names. Hence we exclusively focus on domains containing them. For each combination of a first party and potential CDN, we then again resort to manual checks to determine if this is, in fact, a CDN. In many cases, this is straight-forward based on the involved domain names, such as `soufun.com` and `soufunimg.com`. In checking individual domains to see if they are a CDN, we also observed a notable trend, namely the fact that accessing the CDN

directly (i.e., `http://sitecdn.com`) would redirect us to the main site. Therefore, we augment our manual analysis by leveraging this observation to automatically check, whenever a possible CDN is discovered, if accessing it redirects us to the main site. If that is the case, we mark it as the site's CDN without further manual review. By combining both techniques to identify same-party candidates, we in total identified 2,175 site pairs for further checks, out of which 1,146 are operated by the same entity (across all crawls). Overall, all manual efforts combined took approximately eight person-hours, and we make our results available as part of SMURF [30], which we discuss in Section VIII-C. We augment our list with same-entity entries from the most up-to-date list used by webXray [14] as available in the Internet Archive. Doing so allows us to find 133 additional same-party relations. Contrarily, webXray's list does account for 1,096 of our 1,146 found connections meaning that it alone does not suffice for our purposes.

1) Threats to Validity: The manual clustering approach we chose naturally suffers from a certain limitation in missing sites that belong to the same party. One prominent example is Alibaba, which uses `alicdn.com` on a number of their properties. Notably, though, the combination of the individual sites (e.g., `alibaba.com` or `alipay.com`) does not occur often enough to qualify for the first check we perform. On the flip side, given that `alicdn.com` is not exclusively used on `alibaba.com`, the second check also fails to detect the relation. Luckily, these rather obvious relations of popular sites are picked up by webXray's list [14]. To understand the impact of this on our heuristics (i.e., without the webXray list), we conducted a manual spot check. Based on the total of 183,028 inclusion relations (between different eTLD+1s) we gathered in our first crawl, we could assign 1,434 pairs to be originating from the same party. Of the remaining 181,594, 70,973 could be trivially shown to not originate from the same party; merely because they were included through services like Google's Ad services, for which we are confident to know all related domains[3]. Of the remaining 110,621 pairs, we randomly sampled 1,000 and manually checked if they were of the same party. In doing so, we found that only 24 pairs were actually from the same party. Given our approach of removing the trivially obvious different parties before this sampling, we are confident that our approximation of same-party relations is reasonable. Thus, while our approach may still overestimate the number of third parties for any given Web site, it is much better compared to approaches merely based on the eTLD+1 (as we show in Section V).

C. Holistic View on Disconnect from First Party

Besides having a clear understanding of which hostnames belong together, we want to be able to quantify how (un)related a particular party is to the first party. Prior work [10] used the longest chain of inclusions to measure *implicit trust*; instead, we use the shortest path observed in *any* inclusions from a given party to ascertain its disconnect from the first party.

[3]we conducted a spot check of around 1,000 domains classified as non-Google domains and could not find a single false positive

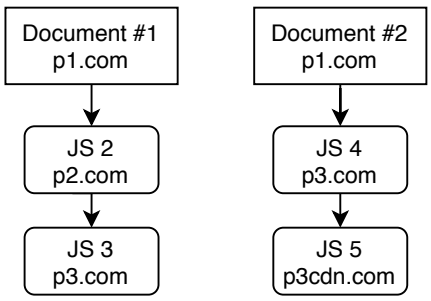

Fig. 1. Example inclusion trees

Figure 1 depicts our running example of two inclusion trees spanning four scripting resources. We use it to introduce concepts that allow us to quantify the disconnect to the first-party developer on the level of parties. We first focus on the left-hand side of the graph. Here, the Web document from p1.com (our first party) includes a script resource JS 2 from p2.com, which in turn includes JS 3 from p3.com. Judging merely on this inclusion chain, p3.com seems to be disconnected from the first-party developer. However, looking at the right-hand side document, we find p1.com directly includes JS 4 from p3.com, meaning there is actually no disconnect. If we now turn our attention to the inclusion of p3cdn.com, we see that it is never included directly by the first party. Considering the eTLD+1 notion, we would flag p3cdn.com as a delegated party, as its inclusion is merely a product of the delegated capabilities of script inclusion to p3.com. However, if we infer that those two sites are in fact to be considered to be the same party, then we would report that p1.com never includes a delegated party. This example highlights the necessity for an improved notion of a same party as well as a means of investigating the shortest chains to a party. In our analysis, we conduct this aggregation for all observed documents belonging to a common root node; e.g., if we find Facebook iframes on another site, we attribute all inclusions within that iframe to Facebook.

For every party inside a given site, we can now calculate the smallest number of other third parties that are scattered along our inclusion chains for any of the hostnames that we can associate with the given party. This allows us to holistically quantify their disconnect from the first party and a delegated party can then be defined as a party for which this number is greater or equal to one.

V. Measuring Disconnect on the Web

Given that prior work has relied on longest chains of inclusions and used an eTLD+1 as the separator between parties, in this section, we study how this notion compares to ours, which relies on shortest paths to a party and the more fine-grained eSP notion. The data is based on the first snapshot of our crawls from January 13, but the results generalize, as Table VI in the appendix confirms.

TABLE I
SITES WHICH HAVE AT LEAST GIVEN NUMBER OF INVOLVED PARTIES IN LONGEST CHAIN

parties	1	2	3	4	5	6	7
eTLD+1	7,643	6,589	3,578	786	137	50	1
eSP	7,628	5,124	1,451	199	19	5	0

TABLE II
LEVEL OF DISCONNECT BETWEEN THIRD PARTY AND FIRST PARTY BY LEAST NUMBER OF THIRD PARTIES ALONG ANY INCLUSION CHAIN.

parties	1	2	3	4	5	6
eTLD+1	7,643	5,807	2,215	315	43	19
eSP	7,625	3,853	750	49	6	2

A. eTLD+1 vs. eSP

In this particular experiment, we want to investigate how our notion of eSP influences the number of different parties that jointly contribute to one inclusion chain. Considering our running example shown in Figure 1, the left branch involves two parties. For the right-hand side, depending on the notion of a party, we have two (for the site notion) or one (for the eSP notion) party involved. We disregard the first party, which means that this number directly reflects the amount of different third parties along any chain in the application. In the example, though, as we are counting *most* involved parties in any chain, the document counts as having two code contributors regardless of the used party notion.

Table I depicts the number of sites and the corresponding number of code contributors involved in *any* inclusion. *eSP* counts the number of distinct code contributors according to our notion of an extended Same Party, whereas *eTLD+1* shows the number according to prior works [2, 10, 22]. Comparing the two notions, a clear difference becomes apparent, which highlights the need for our refined notion. Our results show that for our definition of an extended Same Party 7,628 sites (7,643 for eTLD+1) have at least one additional party from which code is included (shown as 1). This number is in light of our successful detection of 1,146 same-party relations and the 133 relations extracted from webXray. Nevertheless, the majority of these sites also included actual third-party content, explaining the comparatively low difference in numbers.

We find that 5,124 sites have pages on which a directly included third party includes resources from another third party (indicated by having two involved parties, 6,589 for the eTLD+1 notion); i.e., 5,124 sites show a delegation of trust in the longest observed inclusion chain. This is a significant difference of 1,465 sites (18% of the sites with any JavaScript), which would have incorrectly classified as containing delegated inclusions if we had relied on eTLD+1. Hence, we find that our eSP notion provides a significantly better display of inclusion practices in the wild. However, in the following, we highlight the necessity to holistically investigate a site and consider all inclusions in all documents to arrive at a meaningful understanding of trust disconnect.

B. Longest vs. Shortest Path

While investigating the extreme chains provides us with very interwoven interactions among multiple parties, it does not yet allow us to reason about the disconnect between the first-party developer and the code contributor that, in the end, runs their code in the first-party site. To provide a more meaningful notion of such a disconnect, we resort to finding the shortest path to any party that runs code in the site, as in having the least amount of other third parties contributing to the inclusion of a script from the given party as introduced in Section IV-C. In particular, we count how many third parties are *between* the first and the final third party. As discussed in the previous section, this analysis is conducted on all documents belonging to a given root node (Tranco list entry).

Table II depicts our findings with the number of sites for which we can find at least one representative of the party, which depends on the number of other third parties and no other representative being included in a shorter path. We find that 7,625 sites for our extended Same Party notion and 7,643 sites for the eTLD+1 notion include at least one third party and do so directly without the involvement of any other party (meaning they are directly connected but are not the first party, i.e., have a level of disconnect equal to one). What is more, on 3,853 sites code originating from an *implicitly* trusted party is included; i.e., an explicitly trusted third party includes code from somewhere else, denoted as a *delegated party*. Moreover, we find that 750 sites include code from parties to which trust has been delegated twice (i.e., a delegated party included code from yet another party). Finally, 49 sites have at least three levels of trust delegations, and two sites have five.

Our comparative (longest chains with site notion vs. shortest paths with eSP notion) analysis indicates that while sites tend to exhibit highly interwoven trust chains *somewhere* in their pages, considering the holistic view on the code disconnect within a Web site, which we could gather by favoring depth over breadth, provides a much clearer picture. When we compare the trust approximations provided by the longest chain and the site notion with the shortest path and the eSP notion, which account to 6,589 and 3,853 respectively, we can see that **2,736** (34% of our dataset) sites do not suffer from the dangerous pattern of including parties in a delegated fashion.

And while we cannot reproduce the findings of prior work or retroactively apply our methods to their data, our results indeed illustrate that for the current web models of trust disconnect would be heavily skewed when resorting to the longest path and eTLD+1 notion.

For the following analyses, we rely on our established notions; i.e., both for separating parties from each other as well as to reason about delegated or direct inclusions.

VI. IMPAIRING CONTENT SECURITY POLICY

Equipped with our improved notion of parties and third parties' disconnect from the first party, in this section, we quantify the impact of third parties on a site's ability to deploy

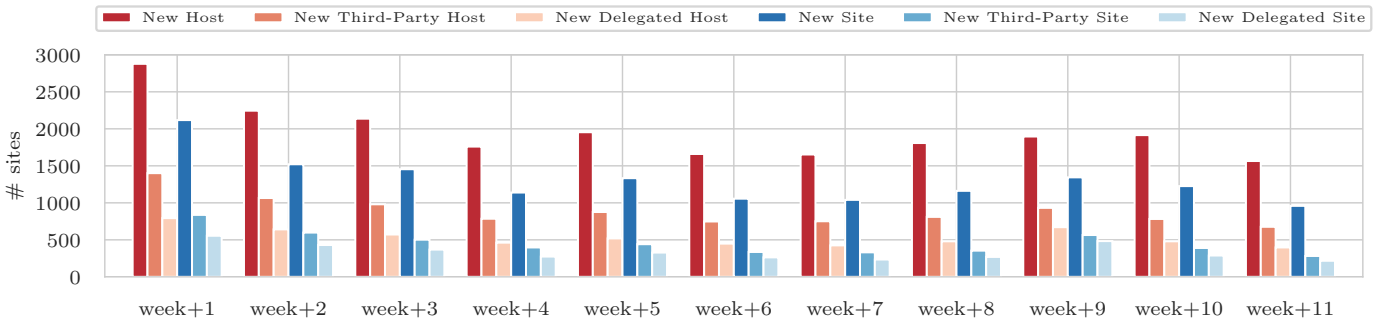

Fig. 2. Stability of included hosts

CSP securely. CSP is primarily meant to protect against XSS. This protection mechanism is undermined if a policy requires the `unsafe-inline` and `unsafe-eval` keywords, which are necessary if inline script or event handlers are used, or strings are transformed to code through `eval`, respectively. Orthogonally, while host-based allowlists are known to be prone for bypasses [3, 39], they are nevertheless recommended to constrain the sources from which developers can include code in the presence of nonces and `strict-dynamic` [16]. This implies that fluctuations in included hosts either break an application or force the first party to allow wildcards such as `https://`. To ease the burden on maintaining a host-based allowlist, sites can also decide to deploy `strict-dynamic`; this, however, is contingent on all included code being compatible through programmatic addition of script elements.

In this section, we investigate how these three aspects of CSPs are impacted by first, third, and delegated parties. Naturally, incompatible code does not technically prevent a first-party from deploying a sane CSP. However, any incompatibility means that specific parts of the site will no longer work, threatening, e.g., functionality or monetization. In particular, we consider this analysis to be an important first step in analyzing to what extent third parties may be involved in the lacking [1] and insecure [39] CSP deployments found throughout top sites, which we further investigate at the end of the section.

A. Host-based Allowlists

As prior work has shown, coming up with a host-based allowlist for CSP is a tiring process, frequently leading to operators simply adding the `*` source expression to avoid breakage [27]. While the insecurity of such a policy is obvious, we here aim to understand to what extend fluctuations in included hosts play a role in first party's struggle to achieve a secure *and* functional CSP.

To keep a policy functioning without causing breakage, it is necessary to allow content from all those hosts which are included. Using a host-based CSP, this can be achieved by individually allowing each host or using a wildcard to allow all hosts belonging to a common eTLD+1 (`*.domain.com`). Allowing all subdomains, however, may expose a site to additional risks. A known bypass to the security of a CSP is to allow sources which host a JSONP endpoint [39]. Naturally, allowing any subdomain of a given domain increases the chances of such an endpoint being allowed. As examples show[4], such endpoints are often contained on subdomains of widely-included domains, e.g., on `detector.alicdn.com`.

Hence, it is desirable to keep the list of allowed hosts as small as possible and resort to allowing all subdomains only if need be. Fluctuations in the included hosts, though, may result in breakage in such cases. Figure 2 shows the stability of the involved hosts and sites over time. In particular, for each week, it shows how many sites include code from hosts they had not previously used (new host) and how many sites introduced code from other eTLD+1s, requiring changes to the host-based allowlist, or allowing the entire subdomain-tree of the new eTLD+1s. In addition, the graph shows the numbers broken down to those hosts/sites which are mandated through third parties; in particular, *New Third-Party Host* refers to the case where a third party introduces a new host, and *New Delegated Host* refers to a third party adding a host from yet another third party. Note that if a first party includes content from a given host, and the third party also includes content from the same host, this is not counted towards third-party inclusions.

In total, 5,442 sites added a new site at least once through our experiment (relative to the sites they included in the first snapshot). 2,977 did so because a third party included content from a new host; of these, 2,272 had delegated inclusions, i.e., a third party introduced code from another third party's hosts. Hence, *55%* of all sites that need to update their CSP (by adding an entire new eTLD+1 and its subdomains) would need to do so because of at least one third party or suffer from functionality breakage. Looking at the trend, we find that while in the first week, over 2,000 sites still introduce content from entirely unseen eTLD1+s, the number goes down to approx. 1,000-1,500 for the following weeks. Interestingly, there is no clear downward trend in the data, implying that even in a longer experiment, we would have observed similar numbers for the following weeks. Notably, the introduction of sites is necessitated by third parties in approx. one-third of all cases; most of these are related to the introduction of sites that do not

[4]https://github.com/zigoo0/JSONBee/blob/master/jsonp.txt

originate from a previously seen third-party (New Delegated Sites). Since these numbers do not contain third parties which are added by the first party, this implies that third parties often add previously unseen parties to a site, requiring the first party to update their CSP with disconnected parties.

Considering that the addition of hosts occurs even more frequently than the addition of new sites, a site operator might resort to allowing all subdomains of a given eTLD+1, so as to avoid having to allow new hosts of the same eTLD+1 in the next week. Notwithstanding the danger of allowing JSONP endpoints, having a CSP that contains entries which are no longer needed violates the principle of least privilege. Operating under the assumption that a site operator would have wanted to keep their site functional and merely added all eTLD+1s that were needed at least once in the 12-week period, 5,544/6,050 would contain unnecessary sites in their CSP at the end of the experiment. That is to say, the vast majority of sites would violate the principle of least privilege. Of these 5,544 sites, 4,135 would have at least one third-party-included (i.e., delegated) host in its overly permissive allowlist.

Given this data, it seems hardly feasible to keep an individual site's host-based CSP up-to-date. Not only is it necessary for many sites to add required hosts or sites to their CSP, but at the same time, a site operator regularly has to assess if their CSP is not too overly permissive, and remove non-needed entries. More than half of the sites that required adding a new eTLD+1 to their CSP were sites with changes initiated by third parties. Similarly, 4,135/5,554 (74%) sites would have to remove a third-party site at least once during the 12 weeks to keep their policy as strict as possible. Naturally, if an operator decides to only allow specific hosts instead of entire sites, there are more changes necessary. Hence, we find that third party induced changes to the allowlists play an important role in the maintenance cost for site operators, requiring significant overhaul on a weekly basis.

1) Categorization of Culprits: To understand this constant influx of newly included sites, we analyze how these new inclusions support the first party. To that end, we utilize Webshrinker [38] to categorize each of the eTLD+1s from which new JavaScript was included throughout our experiments starting from the second week. In particular, we resort to the label with the highest-ranking score to flag an eTLD+1.

Table III shows the most prevalent categories for our entire analysis period, both in terms of inclusions that were caused by either party, as well as for third parties in particular. Not surprisingly, we find the biggest culprit to be IAB3 (Business), which overlaps with IAB3-11 (Marketing) and IAB3-1 (Advertising); i.e., most of the newly introduced sites are related to advertisement. Considering only eTLD+1s that were added by third-party code, 1,325 sites had a least one new inclusion from an ad-related site. The second large cluster of introduced eTLD+1s is related to technology & computing; this category subsumes services that offer email (e.g., newsletter delivery) or chat integrations. IAB25-WS1 contains sites like `gstatic.com` or `nr-data.net`, i.e., it subsumes cases of content distribution. Overall, we can say that the ad ecosystem appears to be the driving factor behind the influx of new eTLD+1s in most sites. However, there exist also other fundamental building blocks included in modern Web sites, which cause the introduction of new sites throughout our experiments.

TABLE III
CATEGORIZATION OF SITES ADDED OVER TIME

Category	affected sites all	affected sites only TP
IAB3 (Business)	2,864	1,325
IAB19 (Technology & Computing)	2,790	725
IAB25-WS1 (Content Server)	1,798	813
IAB25 (Non-Standard Content)	889	284
IAB14 (Society)	758	208

2) Code Location Drift: Naturally, the vast influx of new hosts and eTLD+1s into a significant fraction of our investigated sites begs the question of whether or not this is an artifact of recent findings of Vastel et al. [35]. They were able to show that ad providers are frequently using new hosts to evade filter lists. In particular, we could see that the largest culprits behind the influx of hosts appear to be ad-related. To understand whether this observation is merely a side effect of the evasion techniques performed by ad providers or whether there is actual new code being included from these newly introduced hosts, we set out to analyze whether we can find evidence for code drifting from one already included host to a freshly included one across our analysis periods.

For each site, we checked all scripts included from newly introduced hosts. We then checked all previous snapshots to determine if the same script (based on its SHA256 hash) was previously found on a different host *and* included by the site. In doing so, we found that over the entirety of our dataset, 220 sites had at least one script moved to a new host. More notably, this only affected 814 scripts over all snapshots. Putting this into the perspective of 10,271,782 unique scripts just in the final snapshot, the number of newly added hosts as part of code drift (w.r.t. hash-equal scripts) is minuscule.

Overall, this means that only a tiny fraction of newly introduced hosts can be explained by code drift. On the contrary, we argue that when we see new hosts being added to a site that this also means that new code is added. We can conclude that the vast majority of changes cannot be attributed to filter list evasion techniques and that the artifacts that we measured highlight the general evolution of sites over time, which is a major hindrance for host-based allowlists.

B. Necessary Unsafe Keywords

Next to the struggle of maintaining host-based allowlists, a second major issue to the security of a CSP is the usage of compatibility keywords, namely `unsafe-inline` to enable inline scripts and event handlers, as well as `unsafe-eval` to allow the usage of `eval`. While the former is *always* a serious issue, `eval` has its use-cases, e.g., for local code caching and execution. However, its usage has been discouraged by prior

TABLE IV
SITES WHICH NEED TO USE UNSAFE DIRECTIVES

	unsafe-inline			unsafe-eval
	total	handler	script	*total*
mandated by any	7,667	6,879	7,650	6,334
mandated by first party	7,643	4,972	7,618	4,424
mandated by third party	6,041	5,977	3,601	4,911
- only third party	24	1,907	32	1,910
- multiple third parties	4,573	4,446	1,663	2,943
- delegated parties	1,299	1,251	287	946
- only delegated parties	0	14	0	51

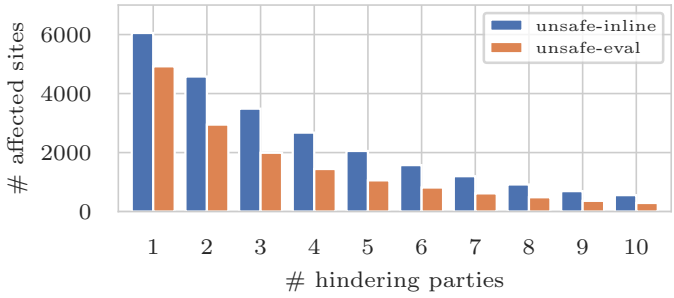

Fig. 3. Sites that require unsafe keywords by multiple third parties

works [26], and the CSP authors' choice to disable by default underlines its security impact.

Given these insights into the keywords we want to avoid in a CSP, we conduct a hypothetical what-if analysis, assuming that all first-party developers wanted to deploy such a policy without any compatibility modes and determine to what extent the different stakeholders provide code that is incompatible with such a policy. To that end, we need to measure when a script uses `eval`, which automatically necessitates `unsafe-eval`. For `unsafe-inline`, we need to monitor access to the DOM through APIs like `document.write` and `innerHTML`; however a mere access is not yet a compatibility issue for CSP. Rather, this behavior only causes issues when used to write additional script tags, or when defining HTML-based event handlers. To measure the behavior of the scripts divided by our different stakeholders and analyze their interaction with security-sensitive functionality, we resort to in-browser hooking of the APIs in question. Together with with our reporting mechanism, these hooks allow us to store the execution trace for each API access and attribute each call to a party. While there are ways for sites to detect such hooking, we do not believe this to be a major threat to validity (in the worst case, it provides us with lower bounds).

Table IV shows the results of our analysis concerning the functionality used by first-, third-, and delegated-party code, which, in its current form, requires either one of the insecure directives. Since `unsafe-inline` is required if *either* inline scripts or HTML event handlers are used, we show those numbers both separately and in sum.

a) `unsafe-inline`: For `unsafe-inline`, we find that 7,667 of our 8,041 analyzed sites have code constructs that require this insecure directive, with the vast majority requiring it due to the usage of inline script elements. Out of those, 7,643 would have to deploy `unsafe-inline` anyways due to their own incompatible code (7,650 due to inline scripts, and 6,879 due to event handlers). Besides, we find that 6,041 sites make use of third-party code, which requires `unsafe-inline` to work. Therefore, even if a first party could rid itself of event handlers and inline scripts, those sites would be hindered by third parties from deploying a CSP without the unsafe keyword. While this seems like a big ask, it is feasible for first parties to deploy a nonce-based policy, enabling them to allow all their inline scripts; event handlers, however, cannot be allowed this way. Considering only those 2,671 sites with first-party inline scripts, but without first-party event handlers (not shown), we find that third parties would induce incompatibilities in 1,903 (71%) of them, which prevents them from a sane CSP even if the first-party made all their code compliant.

Hence, the logical next step in securing a first-party site would be to convince the included third parties to update their code to no longer require `unsafe-inline`. As the table shows, 4,573 sites have more than one third party, which hinders them from deploying CSP without `unsafe-inline`. Additionally, 1,299 sites are hindered through delegated parties, i.e., contributors with which they have no direct relation. Figure 3 shows how many sites have incompatibilities with a sane CSP that stem from how many third parties, i.e., how many parties would need to change their codebase to allow for a breakage-free CSP without unsafe keywords. Unfortunately, more than 2,000 sites (25% of our dataset) would require the cooperation of at least five other parties. There also exists a rather long tail involving still more than 500 sites with ten or more contributors.

b) `unsafe-eval`: For `eval`, the results differ slightly. Overall, 6,334 sites could not deploy a policy without `unsafe-eval` without breakage. In this case, 4,424 (70%) are making use of `eval` in first-party code, and 4,911 (78%) through third-party code. Thus, 1,910 sites cannot deploy a policy without `unsafe-eval` exclusively because of third parties. Even if all sites removed `eval` from their own code base, 2,943 would have to convince more than one third party to do the same (as shown in Figure 3). Similar to `unsafe-inline`, we observe a long tail as well, requiring 292 sites to convince more than ten third parties to remove their usage of `eval` to rid the first party's CSP of `unsafe-eval`.

c) Categorization of Culprits: Combining the blockage through inline scripts, event handlers, and usage of `eval`, 6,377 sites include third parties that mandate either unsafe keyword. To better understand how these parties relate to the business needs of the first party, we categorized all third parties that require compatibility modes, again relying on the results from Webshrinker. Here, we use the first category that is associated with any of a party's eTLD+1s. Table V show the results of this categorization, indicating how many sites are mandated to use `unsafe-inline`, `unsafe-eval`, and

TABLE V
TOP CATEGORIES OF PARTIES THAT REQUIRE UNSAFE KEYWORDS

Category	unsafe-inline	unsafe-eval	either
IAB3 (Business)	3,257	2,235	3,530
IAB19 (Tech. & Comp.)	1,918	1,472	2,717
IAB14 (Society)	1,340	76	1,372
IAB25-WS1 (Content Server)	598	895	1,236
IAB3-11 (Marketing)	633	536	794

at least one mode per category. We find that business is the most prevalent category for both compatibility modes, again likely relating to advertisement. While IAB19 is again second (as it was for inclusion of remote content), IAB14 (Society) is third-most prevalent overall, yet primarily for its usage of inline scripts/event handlers. Taking a closer look at the data, we find that this is caused Twitter, whose scripts from `platform.twitter.com` alone are responsible for 1,294 sites that require `unsafe-inline`. The results for this analysis paint a similar picture for the one of included host, confirming the long-held beliefs that the advertisement industry hinders CSP deployments with empirical evidence.

It is worth noting, though, that simply blaming the ad industry is unfair. While their code contributes to many incompatibilities, we find that removing the categories related to advertisement, 6,213 sites contain other dependencies that require the unsafe keywords. This highlights that simply convincing the ad industry to programmatically add scripts and event handlers as well as stop relying on `eval` does not suffice, but rather a coordinated effort of virtually all third-party content providers is necessary to remedy the situation.

C. `strict-dynamic`

As indicated in Section VI-A, the parties and hosts included in the sites we analyzed fluctuates significantly over time. This observation unveils issues of approaches such as CSPAutogen [24], which rely on a fixed set of hosts to generate the CSP. As outlined in Section II-B, `strict-dynamic` was developed to alleviate this burden, enabling trusted scripts to programmatically add additional scripts. Specifically, this means that all scripts must be added via the programmatic creation of script elements (through `document.createElement`) and the programmatic addition to the DOM (e.g., through `element.appendChild`). Based on our crawl, in which we collected information about how scripts are added at runtime, we find that only 1,414/8,041 (18%) sites would be hindered from properly using `strict-dynamic` due to third parties not adhering to this paradigm when adding additional scripts. Unfortunately, using `strict-dynamic` mandates the usage of nonces or hashes, which in turn means `unsafe-inline` is ignored. And while it is feasible to attach nonces to inline scripts or allow them through their hash, event handlers cannot be allowed in this fashion. The only solution for these issues is to use `unsafe-hashes` [37], yet another compatibility mode. Looking back at Table IV, specifically at third-party induced inline event handlers, 5,977 sites could not use `strict-dynamic` without losing the functionality provided by these handlers. We only find 1,894/8,041 sites without third-party event handlers and where third parties only programmatically add scripts. Hence, the remaining 6,147/8,041 could not deploy `strict-dynamic`.

To conclude our hypothetical scenario, we have seen that even if developers would want to get rid of the compatibility modes, for 6,041 and 4,911 sites, respectively, they would need the cooperation of at least one code contributor, and most likely even multiple ones. We have seen instances in which those contributors are even included over trust delegations, begging the question of whether there is even an incentive for these parties – given the lack of a direct business relation – to change the codebase. This inability imposed by the sites' business needs is particularly problematic given recent ideas of requiring security features, e.g., a strict CSP, to allow the site to access newly introduced browser APIs [19] or even disallow existing APIs to be used given the lack of the respective security feature. While such changes would force developers to act and deploy security mechanisms, our analysis shows that this would still require the cooperation of other parties and can only be tackled by all the stakeholders involved in the Web platform.

D. Real-World Impact on Deployed CSPs

To understand if our hypothetical scenario can be founded by empiricism, we now turn to analyze the policies which we encountered during our crawls. Out of the investigated 8,041 sites we found 1,052 to be using a CSP with either `default-src` or `script-src`, meaning that they make use of CSP's functionality to restrict which scripts end up running within their sites. Out of those 1,052 sites, 1,006 incorporate `unsafe-inline` without nonces or hashes. We found that 707/1,006 sites have third parties that introduce inline scripts. Notwithstanding necessary changes to the first-party code, this means those sites are bound to use `unsafe-inline` to preserve functionality due to third-party code. Confirming our insights from Table IV, all of those besides one site, though, also have inline scripts in the first-party code. In addition, 860 sites make use of `unsafe-eval`. Of those, 540 are partially hindered due to third parties, and 174 solely due to third parties, i.e., first-party code did not use eval. These results not only confirm that over 95% of policies are insecure [39], but more importantly that between 63% (for `unsafe-eval`) and 70% (for `unsafe-inline`) of all sites have third parties that require the unsafe keywords, making policies trivially bypassable.

VII. IMPAIRING SUBRESOURCE INTEGRITY

An orthogonal threat to XSS is the compromise of widely-included resources. By default, external scripts are only referred to by their URL. Hence, if an adversary manages to compromise the network link to such an external resource or the server hosting it, they can freely change the content to include malicious code. As outlined in Section II-C, SRI is

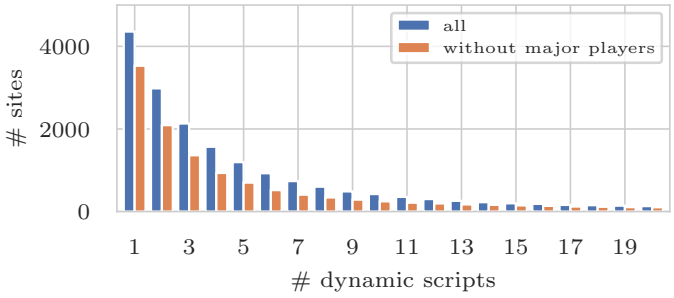

Fig. 4. Sites with Dynamic Scripts

meant to provide a site with enhanced control over external scripts, through the unambiguous specification of the script code's expected hash. Arguably, this may not be desired by the third party (e.g., to avoid breakage when a new version of an advertisement library is released). However, for the first party, this is the only viable option to ensure their included resource has not been changed in a malicious fashion.

As we have shown in Section V, Web sites extensively rely on third parties to provide functionality. However, this in turn widens the attack surface of the including application, since it needs to rely on *all* included parties remaining uncompromised. To ensure that a third-party compromise does not have an adverse effect on their application, it is in the first party's best interest to pin as many third-party scripts as possible. In the following we, use our eSP notion to identify third-party scripts and examine to which degree SRI deployment is feasible to mitigate compromised inclusions.

A. Feasibility of SRI Enforcement

Since our crawlers collect not only the links to included resources, but also their content, we can ascertain which scripts are unstable, i.e., return different content for subsequent requests. Specifically, we analyzed our first snapshot from January 13[5], and extracted all those scripts that within a single crawl (which takes less than 12 hours) returned different content under the same URL. Figure 4 shows the number of sites that have at least one third-party script with changing content. We find that this pattern occurs frequently, with 4,358 sites including such an instable script. The graph also indicates that there is a large body of sites on which there are numerous dynamic scripts. In the long tail, there are 200 sites with 15 such inclusions, and 132 with at least 20.

Table VII in the appendix shows the 20 most commonly occurring scripts. We compared the different versions of the scripts and found that the seven scripts related to Facebook merely differ in a comment that indicates the JIT compile time. In this case, the change does not alter functionality, and the most widely used script alone affects 789 applications, which cannot SRI-pin the included SDK script. Similarly, for Amazon, we found that their ad script was unchanged except for a recompilation, which caused a new comment with a timestamp as well as randomized identifier names. For the files related to Google (Youtube, Ad Services, their API platform), the observed changes were related to versioning; in these cases, the included script would merely add the latest version of another script. While this makes sense from a functionality point of view, as site operators never have to update their inclusion URLs, it nevertheless makes SRI deployment impossible. Arguably, though, the level of security for these high-profile companies may suffice to ensure the integrity of the scripts; in addition all of them were loaded over HTTPS, rendering a man-in-the-middle attack infeasible.

To also assess the effect of less prominent contributors on SRI feasibility, we exclude *any* script from an eTLD+1 from which the top 20 changing scripts originate. Figure 4 also shows the result if we disregard scripts from those hosts. Here, we find that 3,530 sites include at least one scripting resource which changed its content within one crawl. This means that 828 applications could enable SRI if only the top players would fix their content. Yet, the majority of sites could still not deploy SRI for all their third-party inclusions, having to hope their included parties do not suffer a compromise.

1) Programmatically Enforcing SRI: When looking at the average number of different versions for a script under the same location, we found around twelve distinct versions within a snapshot. While SRI does not permit to have multiple integrity tags, assuming that this set of script variants is finite and stable, one could resort to programmatically trying to add a given script with all its possible hashes until one inclusion succeeds; essentially allowing the twelve different hashes for a dynamic script.

Looking in more detail at the findings from Figure 4 for such domains with five dynamic scripts, 1,197 fit that bill. Hence, those applications would, on average, need to incorporate around 60 different hashes to account for all possible script hashes which we could observe. There is a particularly long tail with 132 applications requiring around 240 different hashes solely to allow for the dynamic scripts. To even attempt curating such a list, the first party would need to be informed about any changes in third-party scripts in real-time. This also mandates that the first party can vet any such change in real-time and subsequently incorporate the changes (be it additions or deletions). Given that this seems to be a highly unlikely scenario, we find that SRI cannot be used to constrain the behavior of third-party script resources on the current Web.

2) Undermining SRI Through Additional Inclusions: Besides non-stable script resources, a second hindering factor for comprehensive, site-wide SRI deployment for third-party scripts is subsequent script inclusions done by these scripts. As the first party has no control over this subsequent script inclusion process, it also has no way to enforce SRI pinning for the additional scripts. Within our dataset, out of the 7,643, which include code from third-party entities, 3,276 sites only include stable, hence SRI-compatible scripts. Of those, 2,780 are subject to at least one further script inclusion conducted by a pinnable third party. Thus, the overall number of sites, which *hypothetical* could leverage SRI to ensure only security-vetted

[5]We checked other snapshots and the January 13 results generalize

scripts, comes down to merely 496.

B. SRI in the Wild

As the DevTools Protocol provides no API that allows for retrieval of information about a specific script's SRI usage we need to obtain this information from the DOM directly. We utilize a MutationObserver, which allows us to record changes on any DOM nodes during execution. We register this observer before the page is parsed, allowing us to capture all script additions. Whenever we observe that a script with the `integrity` attribute is introduced, we log this to our database.

While SRI is used on a total of 1,562 sites, we found that 626 of these only used SRI to pin popular libraries, virtually all of which were jQuery or Bootstrap (in line with what Chapuis et al. [4] found). We attribute this fact mainly to the inclusion advice on the homepages of both projects, which supply HTML code snippets already, including the `integrity` attribute. Since these applications exclusively use SRI on library inclusions, we suspect that developers merely use this mechanism by accident. For other libraries (as indicated by `retire.js`), we could not find such hints which explains their lack of SRI pins.

Of the remaining 936 sites, another 22 only pinned first-party resources, which, in terms of content restriction, does not provide any security improvements for the first party. If an attacker would be able to change the pinned resource, they would most likely also be able to change the associated pin, since according to our notion of an extended Same Party, the two resources are both hosted by the same entity. This leaves us with 914 sites on which SRI was actively used to ensure the integrity of a third-party resource.

On 67 sites, we found that an SRI-pinned script included another script without adding an integrity attribute. This subverts the desired effect of SRI, namely to restrict what script code is loaded to what was vetted by the developer. Essentially, an attacker compromising the third party can tamper with the indirectly included script. Furthermore, in 52 cases, the unpinned inclusion belonged to a different party. Essentially, this means that the first-party pins are meaningless, as an attacker can compromise the indirect inclusions. While the lack of understanding of the SRI mechanism has been documented by Chapuis et al. [4], this gives rise to reconsidering how SRI is supposed to operate, e.g., warning a developer through the console that a pinned script conducts an unpinned inclusion.

VIII. Discussion

The Web's success is largely based on a first party's ability to focus on their core business needs, while relying on third parties to provide additional functionality. As we have observed, 95% (7,628/8,041) sites in our dataset rely on at least one third party. However, this reliance on others also means that the first party may be significantly impaired in deploying critical security mechanisms like CSP and SRI. In our work, we underline the community's long-held belief that third parties may be major roadblocks to security header deployment by showing that even if all first parties were to update their own code, the majority would still need to resort to unsafe policies or break third-party functionality. While it is a big ask for the first party, there is little incentive for them to start the process on their own code if third-party functionality necessitates unsafe keywords or induces breakage.

For CSP, we find that given the significant fluctuation in included hosts (both by first and third parties) makes a working, yet secure host-based allowlist infeasible. We could show that 74% of all sites would have to remove sites introduced by third parties from their CSP to keep it as tight as possible. On the flipside, 50% of all sites with fluctuations are required to add sites at least once in our 12-week experiment through inclusions by third parties. Looking at the situation w.r.t. to unsafe keywords, the situation is even more dire; even if first parties were to remove all their usage of eval and inline JavaScript, around 78% of our dataset would require either unsafe keyword because of third parties which rely on them. Importantly, while the notion of `strict-dynamic` as proposed by Weichselbaum et al. [39] could alleviate some of the problems of ever-changing inclusions, a majority of sites with third-party code could not deploy it. This is because of the adverse effect of inline event handlers, which cannot be allowed through hashes or nonces. `strict-dynamic`, however, requires trusted scripts to be allowed through hashes or nonces, meaning a majority of the sites with third-party code could not deploy it without breaking the third-party functionality. Given that sites heavily have to rely on third parties for monetization (be it through ads or reach via social networks), they either have to leave their sites insecure or attempt to rely on other parties to deliver the same service as the incompatible ones. Specifically, for advertisement, to understand if first parties could not simply switch networks, we analyzed all parties which own at least one domain related to advertisement (according to Webshrinker) and were included by at least 1% of our dataset (i.e., 100 sites). This yielded a total of 26 parties, out of which only a single one would not impair the first party in deploying a CSP (i.e., does not require `unsafe-eval` or `unsafe-inline` and does not break with `strict-dynamic`). This was digitru.st, which is not primarily serving ads, but rather helps multiple ad companies synchronize tracking identifiers. This implies that unless large ad networks generally adhere to CSP-compliant coding practices, first parties are left in a no-win situation where they can either have revenue loss or insecure CSPs.

As far as SRI is concerned, we stipulate that while not all resources should necessarily be pinnable, third parties should avoid making them unpinnable through technically unnecessary randomization. In particular, the example of Facebook shows that merely due to comments in the JavaScript file, SRI is rendered inapplicable. As our analysis in Section VII has shown, relying on programmatic cycling through observed hashes of a given external resource is also infeasible due to the large number of third-party inclusions and the average number of different versions returned when requesting the same URL. All this data is only based on a single snapshot, for which

the dynamic nature is even more surprising, given that one crawl only takes around 12 hours. In addition, our real-world analysis showed that SRI-pinned scripts might themselves include additional scripts without SRI; meaning that an attacker may simply compromise the delegated inclusion to attack the SRI-pinning site.

A. Call to Action for Third Parties

Third parties are in a unique position to affect the Web's overall security. As it stands now, they are a major roadblock to the deployment of promising security measures like CSP. Especially regarding the programmatic addition of scripts, the necessary changes in the codebase are minimal. While removing the reliance on event handlers and inline scripts may be significantly more engineering effort, widely-included third parties have a massive amplifier; i.e., they can affect hundreds of sites' ability to deploy CSP. While especially the largest vendors can hardly be compelled to change their functionality to be CSP-compliant, we nevertheless call on them to adopt best practices and lead by example to ensure that a wider-spread adoption of CSP is even feasible. It is worth noting here that we discovered scripts from Twitter to be a CSP roadblock for around 1,300 sites, forcing sites to deploy `unsafe-inline`. Interestingly, their own CSP is nonce-based, which would be incompatible with the code they provide third parties. Hence, if third parties vetted the code they provide to others as much as the one they run themselves, the situation could quickly be remedied.

Furthermore, as far as SRI is concerned, we found that many third parties make their scripts incompatible with SRI for no good reason, e.g., through having a comment with the timestamp in the response. Given that a single compromised CDN poses a grave threat for hundreds of sites, we urge third parties to make every effort to have SRI-compatible scripts.

B. Call to Action for Browser Vendors

Browser vendors move the Web's security forward by implementing novel security mechanisms. For example, while the recent choice of Chrome to make cookies same-site by default [6] may result in breakage, it largely solves problems such as CSRF, XSSI, or Clickjacking. As proposed by Google's Mike West, vendors may consider hiding new features behind the deployment of security mechanisms [19], such as *sane* CSP. While this has its upsides in ensuring that newly developed code that wants to use these new features has to be built in a CSP-compliant fashion, it may also have an adverse affect on existing applications. In particular, if third parties cause compliance issues, the first party cannot use the new features. Hence, while we support such incentive structures, they should be deployed in line with mechanisms such as First-Party Sets [9]. In this way, code from first parties could be allowed to access the new APIs without having to deploy a sane CSP, whereas for third parties access is only granted if a sane CSP is deployed. As Web sites rely on their third parties for monetization, the first-party developers then also have incentives to address their own incompatibilities, leading to an overall more secure Web. Furthermore, given SRI's inflexibility to deal with structurally equivalent code with differing values, browser vendors should consider adopting more relaxed pinning strategies such as the one proposed by Soni et al. [28].

C. Developer Support: SMURF

In an ideal setting, every third-party script could provide a *security manifest*, similar to browser extensions, that declaratively states the component's security impact. In early steps for such a measure, this could, e.g., start by simply stating which domains may be involved in any inclusion or contain proposals for a CSP which supports the component's functionality needs. However, until such a point is reached, we make the most important components of our toolchain available as part of SMURF [30]. With this, developers can analyze their sites to a) understand how inclusions relate to each other, enabling them to untangle which host introduced certain code into their site. In addition, it b) allows to monitor the behavior of all included code to understand if it causes incompatibilities through the usage of `eval` or unsafe script additions. If used during development time, this enables the first party to gain key insights into the security roadblocks which may be caused by including third parties, potentially allowing them to choose another vendor with similar functionality early on.

IX. Limitations

Our work relies on several assumptions about entities on the Web and their incentives. We argue that a party should be defined by the business entity that operates its eTLD+1. For bigger companies, such as Google, there likely are different teams for individual products (such as Youtube or Doubleclick), and hence, multiple teams would have to adjust their coding practices. Nevertheless, a major player such as Google could globally decide to adhere to more compatible coding standards. Furthermore, our heuristics aim at identifying co-occurrence patterns, which we use to manually draw connections between two hostnames if they belong to the same business entity. Even though such an approach is meaningful to differentiate between first and third parties and among different third-parties, it should not blindly be used in other domains. In particular, we do not want the main site and its sandbox domain to be treated the same under the SOP, which would severely threaten the integrity of the main site. Nonetheless, our approach allows us to investigate the number of players (including the first parties) currently hindering a wide-spread adoption of security mechanisms. Another clear limitation of our approach comes from the imperfection of our used heuristics, as outlined in our experiment in Section IV-B1. While our manual analysis (which we assume to be correct given our diligence) does not yield false positives (i.e., flag two domains from different parties as same-party), it does yield false negatives. Prominently, this was the case for Alibaba's CDN and 2.4% of the 1,000 manually sampled inclusion pairs. Hence, we may still incorrectly associate first-party behavior with a third party when relying solely on our heuristics paired

with manual vetting, yet believe this number to be small enough given the error rate derived through our sampling to be reasonably used on their own. For our used notion of an eSP, we have augmented the same-entity pairs found by us with relations from curated lists containing hostname to entity mappings available online [14], to enhance the precision of our results further. Unfortunately, we could show that solely relying on such lists does not suffice for our use-case.

Second, our hypothetical analysis assumes that a first party wants to deploy CSP in the first place. However, the site in question may not have any user-specific data, making an XSS attack less damning. Furthermore, we do not imply that a first party would deploy CSP right away after all of its third-party resources are compatible. As shown by Roth et al. [27], even high-profile sites like Flickr struggle for years with their deployment. However, the task is made even harder if the first party needs to convince its third parties to produce compliant code or switch third-party providers altogether. Similarly, for SRI, we analyze its deployability from the first party's standpoint, who may want to pin all of its resources. This may be in stark contrast to what the third-party provider wants to do, as they may want to disallow SRI usage to enable the seamless rollout of new library versions.

Third, our manual approach to grouping eTLD+1s that belong to the same party makes it hard for others to build on our results. Unfortunately, our attempts at automating this process proved to produce incorrect results (both in terms of false positives and negatives). To allow others to build on our work, we release the labeled set of same-party domains as identified by our work [30].

Finally, our analysis focuses on the top 10,000 Web sites and may not be accurate to represent the entire Web. We favor depth over breadth since our research relies on a holistic view of the checked sites. In addition, the longitudinal aspect means that lower-ranked sites that go offline could significantly skew the data. Nevertheless, it may well be the case that lower-ranked sites include incompatible code less often. We still believe that the process of deploying CSP and SRI (even for lower-ranked sites) would be eased significantly if the third parties of the major sites we analyzed in this paper would adhere to compliant coding practices.

X. Related Work

CSP & SRI: Content Security Policy has been the subject of many studies over the years. In 2014, Weissbacher et al. [40] conducted a longitudinal analysis of CSP deployment, showing virtually no adoption. While follow-up studies from Weichselbaum et al. [39] and Calzavara et al. [3] indicated an increase in CSP deployment, they both independently showed the vast majority of policies are insecure. Most recently, Roth et al. [27] analyzed the historical evolution of CSPs for 10,000 sites, documenting how site operators struggle to secure their CSPs, and often either give up entirely; or fall back to trivially bypassable policies. While attempts have been made to ease the deployment of CSP through automatic generation [24], this has also not caused a significant uptick. Regarding SRI, Chapuis et al. [4] conducted a longitudinal analysis of SRI deployment. While their results indicate an increasing deployment, they more importantly document this is mostly related to widely-used libraries such as jQuery having example code with SRI.

While our analysis cannot conclusively point the finger at third parties, we documented that even in case a first party is willing to make the effort to secure their site with CSP and SRI, they are blocked by third parties. Especially for CSP, we believe this explains the lack of deployment and struggle to deploy a meaningful policy for existing applications. Our work is the first to specifically document the incompatible behavior exhibited by third-party code, impairing CSP deployment. Furthermore, for SRI we highlight that in particular scripts with a high reach (such as Facebook with almost 10% of the sites in our dataset) are often incompatible with SRI due to minuscule, random modifications.

Script Inclusion Practices & Third-Party Analyses: The security impact of third parties has been the subject of research since at least 2012. Back then, Nikiforakis et al. [22] measured the script inclusion behavior of the Top 10,000 Web sites showing that *first-party* inclusion decisions can vastly impact the security of the including site. While this study examines the included resources based on their origin, work from Yue and Wang [41] also investigated the structural properties of dynamically added code. Kumar et al. [10] started to focus more on the structure of such script inclusions and introduced the concept of implicit trust. They furthermore show that a quarter of the top 1 million sites are blocked from deploying HTTPS due to their inclusions. The risks of including outdated libraries were analyzed by Lauinger et al. [11], showing that 37% of the top 75,000 sites include at least one library containing a vulnerability. The dangers associated with malicious links contained in such inclusion chains were highlighted by Arshad et al. [2]. To tackle this problem, they proposed an in browser-solution detecting malicious links, thus, protecting end-users. In 2019, Ikram et al. [8] investigated how often malicious inclusions happen over implicit trust relations in the Alexa top 200,000. Based on their longitudinal analysis, they find that 95% of included parties carry over to the next day. Musch et al. [21] highlighted the threat of third-party caused XSS vulnerabilities and provided a client-side library that automatically mitigates all third-party caused vulnerabilities.

All of these works have made assumptions about parties based on eTLD+1s. While the impact of our findings is limited for the work from Nikiforakis et al. [22], all other works have reasoned about indirect or delegated trust; meaning that our findings indicate their over-approximation of the problem space. We note again that our analysis has revealed that had we taken the old notion of trust delegations (longest chains and eTLD+1), we would have incorrectly flagged **34%** of our dataset as having delegations, where there are actually none. Hence, we highlight the need for a careful analysis of involved parties, and to move away from eTLD+1 as an indicator.

With respect to detecting third-party hosting, Matic et al. [17] proposed to use RDAP information about the resolved IPs

of sites as well as information extracted from the startpages to detect hosting environments. In particular, they investigate if a given site is self-hosted or via a CDN/third party. We experimented with automated clustering based on common names of TLS certificates and the IP ranges of involved domains. However, both approaches yielded imprecise results, both in terms of missing connections (such as `newrelic.com` and their CDN `nr-data.net`) as well as incorrect clusters (such as different parties hosted by Cloudflare). The same restriction applies to the approach of Matic et al. [17]. Hence, while our manual clustering does not necessarily catch all parties correctly, it does not draw false conclusions by grouping domains that do not belong together.

XI. Conclusion

In this paper, we analyzed to what extent first parties, who are willing to change their own code base, can meaningfully secure their sites through CSP and SRI. Based on our new notions of the same party and delegation of trust, we found that third parties are major roadblocks for security. For CSP, they often introduce new delegated hosts, requiring the first party to potentially add entire eTLD+1s to their policies. At the same time, the fluctuation in included parties means that the first party needs to continually remove entries from their CSP to maintain the principle of least privilege. Furthermore, third parties play a major role in necessitating `unsafe-inline` and `unsafe-eval`, both in our hypothetical analysis as well as in the wild. And while updating the host-based CSP could be eased by the deployment of `strict-dynamic`, third parties provide code that is incompatible either due to parser-inserted script addition or through using inline events. Regarding SRI, we find that high-profile parties often randomize minuscule parts of their scripts, which actively hinders pinning. What is more, real-world evidence shows that pinned scripts often include unpinned code from additional, delegated sources, undermining the entire security of SRI.

Arguably, first parties have a significant task ahead in ensuring their own compatibility, especially with CSP. However, even having done so, the majority of them are unable to outsource non-core business needs and deploy security mechanisms at the same time. This leaves them in a no-win situation in which either security can be enforced or functionality preserved. While the former would require them to implement all functionality themselves, choosing the latter leaves them subject to security-sensitive decisions taken by third parties, which themselves face no repercussions when providing code that is incompatible with security mechanisms.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable feedback. The authors gratefully acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[1] April King. Analysis of the Alexa Top 1M sites. Online https://pokeinthe.io/2019/04/04/state-of-security-alexa-top-one-million-2019-04/, April 2019.

[2] Sajjad Arshad, Amin Kharraz, and William Robertson. Include me out: In-browser detection of malicious third-party content inclusions. In *Financial Crypto*, 2016.

[3] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. Content security problems? evaluating the effectiveness of content security policy in the wild. In *CCS*, 2016.

[4] Bertil Chapuis, Olamide Omolola, Mathias Humbert, Mauro Cherubini, and Kévin Huguenin. An empirical study of the use of integrity verification mechanisms for web subresources. In *TheWebConf*, 2020.

[5] Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. Towards usable checksums: Automating the integrity verification of web downloads for the masses. In *CCS*, 2018.

[6] Chrome Platform Status. Cookies default to same-site=lax. Online https://www.chromestatus.com/feature/5088147346030592.

[7] ECMA international. ECMAScript 2017 Language Specification. Online https://www.ecma-international.org/ecma-262/8.0/index.html, 2017.

[8] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. The chain of implicit trust: An analysis of the web third-party resources loading. In *The Web Conference*, 2019.

[9] krgovind. First-party sets. Online https://github.com/krgovind/first-party-sets, 2019.

[10] Deepak Kumar, Zane Ma, Zakir Durumeric, Ariana Mirian, Joshua Mason, J Alex Halderman, and Michael Bailey. Security challenges in an increasingly tangled web. In *WWW*, 2017.

[11] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. In *NDSS*, 2017.

[12] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *NDSS*, 2019.

[13] Timothy Libert. An automated approach to auditing disclosure of third-party data collection in website privacy policies. In *WWW*, 2018.

[14] Timothy Libert. webXray Domain Owner List. Online https://web.archive.org/web/20200604213008if_/https://github.com/timlib/webXray_Domain_Owner_List/blob/master/domain_owners.json, 2020.

[15] Timothy Libert. webXray. Online https://webxray.org/, 2020.

[16] Lukas Weichselbaum and Michele Spagnuolo. CSP - A Successful Mess Between Hardening and Mitigation. Online https://static.sched.com/

hosted_files/locomocosec2019/db/CSP%20-%20A%20Successful%20Mess%20Between%20Hardening%20and%20Mitigation%20%281%29.pdf.

[17] Srdjan Matic, Gareth Tyson, and Gianluca Stringhini. Pythia: a framework for the automated analysis of web hosting environments. In *TheWebConf*, 2019.

[18] William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, and Limin Jia. Riding out domsday: Toward detecting and preventing dom cross-site scripting. In *NDSS*, 2018.

[19] Mike West. Securer Contexts. Online https://github.com/mikewest/securer-contexts, February 2020.

[20] Mozilla Developer Network. Subresource Integrity. Online https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity, March 2019.

[21] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. Scriptprotect: Mitigating unsafe third-party javascript practices. In *ASIACCS*, 2019.

[22] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *CCS*, 2012.

[23] Erlend Oftedal. Retire.js. Online https://retirejs.github.io/retire.js/, 2019.

[24] Xiang Pan, Yinzhi Cao, Shuangping Liu, Yu Zhou, Yan Chen, and Tingzhe Zhou. Cspautogen: Black-box enforcement of content security policy upon real-world onlines. In *CCS*, 2016.

[25] Charles Reis, Alexander Moshchuk, and Nasko Oskov. Site isolation: Process separation for web sites within the browser. In *USENIX Security*, 2019.

[26] Gregor Richards, Christian Hammer, Brian Burg, and Jan Vitek. The eval that men do. In *ECOOP*, 2011.

[27] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. Complex security policy? a longitudinal analysis of deployed content security policies. In *NDSS*, 2020.

[28] Pratik Soni, Enrico Budianto, and Prateek Saxena. The sicilian defense: Signature-based whitelisting of web javascript. In *CCS*, 2015.

[29] Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the web with content security policy. In *WWW*, 2010.

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

[31] Marius Steffens, Christian Rossow, Martin Johns, and Ben Stock. Don't trust the locals: Investigating the prevalence of persistent client-side cross-site scripting in the wild. In *NDSS*, 2019.

[32] Ben Stock, Stephan Pfistner, Bernd Kaiser, Sebastian Lekies, and Martin Johns. From facepalm to brain bender: Exploring client-side cross-site scripting. In *CCS*, 2015.

[33] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. How the web tangled itself: Uncovering the history of client-side web (in) security. In *USENIX Security*, 2017.

[34] Tobias Urban, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. Beyond the front page: Measuring third party dynamics in the field. In *TheWebConf*, 2020.

[35] Antoine Vastel, Peter Snyder, and Benjamin Livshits. Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking. *arXiv preprint arXiv:1810.09160*, 2018.

[36] W3C. Content Security Policy Level 2. Online https://www.w3.org/TR/CSP2/#changes-from-level-1, December 2016.

[37] W3C. Usage of 'unsafe-hashes'. Online https://w3c.github.io/webappsec-csp/#unsafe-hashes-usage, December 2018.

[38] Webshrinker. Webshrinker. Online https://www.webshrinker.com/, May 2020.

[39] Lukas Weichselbaum, Michele Spagnuolo, Sebastian Lekies, and Artur Janc. Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy. In *CCS*, 2016.

[40] Michael Weissbacher, Tobias Lauinger, and William Robertson. Why is csp failing? trends and challenges in csp adoption. In *RAID*, 2014.

[41] Chuan Yue and Haining Wang. A measurement study of insecure javascript practices on the web. *ACM TWEB*, 2013.

TABLE VI

RESULTS OVER OUR COMPLETE ANALYSIS PERIOD FOR OUR NOTION OF AN ESP. WHILE WE CAN SEE THAT THE OVERALL TRENDS REMAIN STABLE THERE APPEARS TO BE A GENERAL DOWNWARD TREND OF THE SPECIFIC NUMBERS. THIS CAN BE MOSTLY ATTRIBUTED TO A NUMBER OF APPLICATION TO WHICH WE CAN NO LONGER CONNECT DURING OUR ANALYSIS PERIOD.

	Date	0	1	2	3	4	5	6	7	8
longest chains	20200113	8041	7628	5124	1451	199	19	5		
	20200120	7943	7530	5039	1421	195	18	4		
	20200127	7897	7487	5028	1411	190	15	5		
	20200203	7872	7457	4988	1400	207	22	6		
	20200210	7513	7086	4632	1219	152	17	6		
	20200217	7867	7438	4990	1370	195	19	5		
	20200224	7761	7339	4873	1314	165	13	5		
	20200302	7728	7295	4835	1291	159	16	6		
	20200309	7910	7496	5047	1429	200	19	6	1	
	20200316	7742	7322	4896	1408	206	22	6	1	
	20200323	7899	7493	5025	1447	216	23	5	1	
	20200330	7855	7463	4991	1415	214	21	4		
shortest path	20200113	7634	3899	762	50	6	2			
	20200120	7538	3818	738	66	6	2			
	20200127	7495	3806	743	69	7	2			
	20200203	7466	3642	713	66	6	2			
	20200210	7097	3497	678	63	7	2			
	20200217	7450	3788	739	65	5	2			
	20200224	7348	3664	702	53	4	1			
	20200302	7304	3430	612	41	6	2			
	20200309	7496	3740	727	67	9	3	1		
	20200316	7322	3691	719	67	7	2	1		
	20200323	7493	3774	724	62	6	2	1		
	20200330	7463	3731	725	70	8	1			

TABLE VII

MOST COMMONLY INCLUDED THIRD-PARTY SCRIPTS THAT CHANGED THEIR CONTENT DURING OUR STUDY. TABLE SHOWS THE NUMBER OF AFFECTED APPLICATIONS AND THE AMOUNT OF DIFFERENT SCRIPT HASHES THAT WE COULD OBSERVE TO BE SERVED UNDER THE RESPECTIVE URL.

script location	affected sites	versions found
https://connect.facebook.net/en_US/sdk.js	789	245
https://www.googletagservices.com/tag/js/gpt.js	641	91
https://securepubads.g.doubleclick.net/tag/js/gpt.js	336	69
https://sb.scorecardresearch.com/beacon.js	276	3
https://connect.facebook.net/en_US/all.js	186	177
https://www.googleadservices.com/pagead/conversion_async.js	142	71
https://pagead2.googlesyndication.com/pagead/show_companion_ad.js	109	37
https://www.google.com/recaptcha/api.js	104	2
https://connect.facebook.net/en_US/fbevents.js	98	3
https://www.googletagservices.com/activeview/js/current/osd.js?cb=%2Fr20100101	95	2
https://pagead2.googlesyndication.com/pagead/js/adsbygoogle.js	57	6
https://www.youtube.com/iframe_api	50	2
https://connect.facebook.net/en_US/sdk.js?hash=42272dd37ca5caf2a2797a1147783a65&ua=modern_es6	50	8
https://www.googletagservices.com/activeview/js/current/osd_listener.js?cache=r20110914	48	2
https://cse.google.com/adsense/search/async-ads.js	48	2
https://c.amazon-adsystem.com/aax2/apstag.js	47	2
https://connect.facebook.net/en_GB/sdk.js	45	114
https://connect.facebook.net/en_US/fbds.js	43	89
https://apis.google.com/js/platform.js	40	2
https://connect.facebook.net/ja_JP/sdk.js	36	84