Measuring and Analysing the Chain of Implicit Trust: A Study of Third-party Resources Loading

MUHAMMAD IKRAM, Macquarie University, Australia
RAHAT MASOOD, UNSW and Data61, CSIRO, Australia
GARETH TYSON, Queen Mary University of London & Alan Turing Institute, United Kingdom
MOHAMED ALI KAAFAR, Macquarie University and Data61, CSIRO, Australia
NOHA LOIZON, Data61, CSIRO, Australia
ROYA ENSAFI, University of Michigan, USA

The web is a tangled mass of interconnected services, whereby websites import a range of external resources from various third-party domains. The latter can also load further resources hosted on other domains. For each website, this creates a dependency chain underpinned by a form of implicit trust between the first-party and transitively connected third-parties. The chain can only be loosely controlled as first-party websites often have little, if any, visibility on where these resources are loaded from. This paper performs a large-scale study of dependency chains in the web, to find that around 50% of first-party websites render content that they do not directly load. Although the majority (84.91%) of websites have short dependency chains (below 3 levels), we find websites with dependency chains exceeding 30. Using VirusTotal, we show that 1.2% of these third-parties are classified as suspicious — although seemingly small, this limited set of suspicious third-parties have remarkable reach into the wider ecosystem. We find that 73% of websites under-study load resources from suspicious third-parties, and 24.8% of first-party webpages contain at least three third-parties classified as suspicious in their dependency chain. By running sandboxed experiments, we observe a range of activities with the majority of suspicious JavaScript codes downloading malware.

CCS Concepts: • **Security and privacy → Web application security**; *Malware and its mitigation*; • **Information systems → Traffic analysis**; • **General and reference → Measurement**; **Experimentation**; • **Networks → Web protocol security**; • **Social and professional topics** → *Malware / spyware crime.*

Additional Key Words and Phrases: Measurement, web of trust, third party resources, javascript, web security and privacy, sandbox, experiments

ACM Reference Format:
Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. 2020. Measuring and Analysing the Chain of Implicit Trust: A Study of Third-party Resources Loading. *ACM Trans. Priv. Sec.* 23, 04, Article 01 (January 2020), 27 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

A preliminary version of this paper, titled "The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading", to appear in the proceedings of the 28^{th} World Wide Web Conference (WWW), San Francisco (2019) [27]. See Sections 1 and 6 for a summary of the new results presented in this paper.

Authors' addresses: Muhammad Ikram, Macquarie University, BD Building, 4 Research Park Drive, Level 2, Macquarie University, Sydney, NSW, 2109, Australia, muhammad.ikram@mq.edu.au; Rahat Masood, UNSW and Data61, CSIRO, Level 5, 13 Garden Street, Eveleigh, Sydney, NSW, 2015, Australia, rahat.masood@data61.csiro.au; Gareth Tyson, Queen Mary University of London & Alan Turing Institute, Mile End Road, London, E1 4NS, London, United Kingdom, g.tyson@qmul.ac.uk; Mohamed Ali Kaafar, Macquarie University and Data61, CSIRO, BD Building, 4 Research Park Drive, Level 2, Macquarie University, Sydney, NSW, 2109, Australia, dali.kaafar@mq.edu.au; Noha Loizon, Data61, CSIRO, Level 5, 13 Garden Street, Eveleigh, Sydney, NSW, 2015, Australia, noha.loizon@data61.csiro.au; Roya Ensafi, University of Michigan, Bob and Betty Beyster Building, 2260 Hayward Street, Ann Arbor, MI, 48109-2121, USA, ensafi@umich.edu.

2471-2566/2020/01-ART01 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the modern web ecosystem, websites often load resources from a range of third-party domains such as ad providers, tracking services and analytics services. This is a well known design decision that establishes an *explicit trust* between websites and the domains providing such services. However, often overlooked is the fact that these third-parties can further load resources from other domains, creating a *dependency chain*. This results in a form of *implicit trust* between first-party websites and any domains loaded further down the chain.

Consider the `bbc.com` webpage,[1] which loads JavaScript program from `widgets.com`, which, upon execution loads additional content from another third-party, say `ads.com`. Here, `bbc.com` as the first-party website, *explicitly* trusts `widgets.com`, but *implicitly* trusts `ads.com`. This can be represented as a simple dependency chain in which `widgets.com` is at level 1 and `ads.com` is at level 2. Past work tends to ignore this, instead collapsing these levels into a single set of third-parties [11, 44]. Here, we argue that this overlooks a vital aspect of website design. For example, it raises a notable security challenge, as first-party websites lack visibility on the resources loaded further down their domain's dependency chain. This potential threat should not be underestimated as errant active content (*e.g.,* JavaScript code) opens the way to a range of further exploits, *e.g.,* Layer-7 DDoS attacks [45] or ransomware campaigns [32].

This paper studies dependency chains in the web ecosystem. Although there has been extensive work looking at the presence of third-parties in general [11, 38, 44], little work has focused on how content is indirectly loaded. We start by inspecting how extensive dependency chains are across the Alexa's top-200K (Section 2). We confirm their prominence, finding that around 50% of websites *do* include third-parties (e.g., content delivery networks (CDNs) such as `akamaihd.net` and ad and tracking services such as `google-analytics.com`) which subsequently load other third-parties to form a dependency chain (*i.e.,* they implicitly trust third-parties they do not directly load). The most common *implicitly* trusted third-parties are well known operators, *e.g.,* `google-analytics.com` and `doubleclick.net`: these are implicitly imported by 68.3% (134,510) and 46.4% (91,380) websites respectively. However, we also observe a wide range of more obtuse third-parties such as `pippio.com` and `51.la` imported by 0.52% (1,146) and 0.51% (1,009) of websites. Although the majority (84.91%) of websites have short chains (with levels of dependencies below 3), we find first-party websites with dependency chains exceeding 30 in length. This not only complicates page rendering, but also creates notable attack surface.

With the above in mind, we then proceed to inspect if *suspicious* or even potentially *malicious* third-parties are loaded via these long dependency chains (Section 4). We do not limit this to just traditional malware, but also include third-parties that are known to mishandle user data and risk privacy leaks [5, 8, 16, 41, 43, 52]. Example threats include the re-identification of users in the anonymised AOL search histories, the Netflix training data that was attacked, and the Massachusetts hospital discharge data [16, 43, 52]. Furthermore, the collection of sensitive data by third parties also had devastating impacts on people's lives. For instance, it was shown that a person discovered his teenage daughter's pregnancy by observing her targeted adverts [8]. Similarly, Gmail was shown to use words from users' emails to target ads, exposing the nature of private correspondence in targeted ads [5].

Using the VirusTotal service [29] API, we classify third-party domains into innocuous *vs.* suspicious. When using a reasonable classification threshold (i.e., VTscore $\geq$ 10, further elaborated in Section 2.2 and 4.1), we find that 1.2% of third-parties are classified as suspicious. Although seemingly small, we find that this limited set of suspicious third-parties have remarkable reach. 73% of websites under-study load resources from suspicious third-parties, and 24.8% of first-party

[1] This is an example (i.e., hypothetical case) to elaborate the (suspicious) resource dependency tree of `bbc.com`.

webpages contain at least 3 third-parties classified as suspicious in their dependency chain. This, of course, is impacted by many considerations which we explore — most notably, the power-law distribution of third-party popularity, which sees a few major players on a large fraction of websites.

This paper expands on our prior work [27]). In this past research we inspected the prevalance of dependency chains in the web. Here, we build on these past findings to focus on *what* activities are undertaken within the dependency chains. Hence, we *sandbox* all suspicious JavaScript programs to monitor their activities (Section 5). We build a sandbox and perform tests executing suspicious JavaScript codes. We find that JavaScript codes loaded at higher levels in the dependency chain ($Level \geq 2$) generated a larger number of HTTP requests. This is worrying as resources loaded at higher levels in the dependency chain are the most opaque to the website operator (*i.e.,* they rely on implicit trust). The activities of these scripts are diverse. For example, we find evidence of first-party websites performing malicious search poisoning activities when (implicitly) loading some JavaScript codes. The most typical purpose of the suspicious JavaScript code is downloading dropfiles[2]. We also observe instances of *very* active JavaScript codes, *e.g.,* the most active (at level 4) downloads 129 files.

In Section 7.1, we summarise our key findings and explore simple solutions that may mitigate the impact of the discussed vulnerabilities (cf. Section 5). We also proceed to highlight some of the key limitations of our work in Section 7.2. We conclude the paper by summarising the reality of a very fragile web ecosystem, confirming that suspicious parties within the dependency chains are relatively commonplace (Section 8). We share all our datasets, experimental testbed code and scripts with the wider research community: `https://wot19submission.github.io`.

2 DATASET AND DATA ENRICHMENT

We start by presenting our data collection methodology, and how we have validated its correctness. This consists of two key parts: (*i*) collecting information about individual websites, such that we can extract their dependency chains; and (*ii*) classifying all dependencies (*i.e.,* third-party domains) as suspicious *vs.* innocuous.

2.1 Alexa dependency dataset

We first present how we have obtained data on website dependencies, and how we construct their dependency chain. This critical first step underpins all subsequent analysis

2.1.1 Data Collection. We obtain the resource dependencies of the Alexa top-200K websites' main pages[3] using the method described in Kumar *et al.* [35]. This Chromium-based Headless [15] crawler renders a given website and tracks resource dependencies by recording network requests sent to third-party domains. The requests are then used to reconstruct the dependency chains between each first-party website and its third-party URLs. Note that each first-party can trigger the creation of multiple dependency chains (to form a tree structure).

Figure 1 presents an example of a dependency chain with 3 levels; level 1 is explicitly trusted by the first-party website, whilst level 2 and 3 are implicitly (or indirectly) trusted. For simplicity, we refer to any domain that differs from the first-party to be a third-party. More formally, to construct the dependency tree, we identify third-party requests by comparing the second level domain of the page (*e.g.,* `bbc.com`) to the domains of the requests (*e.g.,* `cdn.com` and `ads.com` via `widgets.com`). Those with different second level domains are considered third-party. We ignore the sub-domains

[2]Dropfiles are executables (*e.g.,* malware, Exploitkits, Trojans, *etc.*) exploiting the browser to download and execute code without user consent (cf. Section 5.2.3).

[3]We select the top 200K as this gives us broad coverage of globally popular websites, whilst also remaining tractable for our subsequent data enrichment activities.

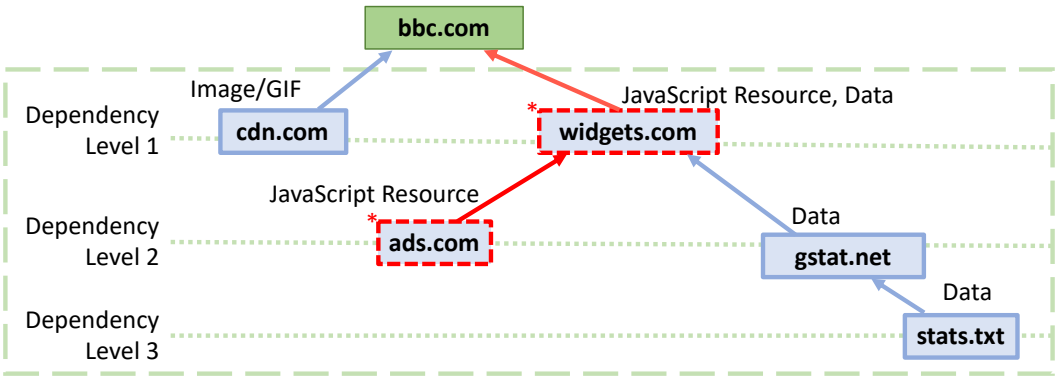


Fig. 1. Example dependency chain of bbc.com. Arrows represent the inclusion of resources with red ones showing suspicious resource inclusion. For instance, ads.com is suspicious, and loaded by widgets.com, creating an implicit line of trust.

so that a request to a domain such as player.bbc.com is not considered as third-party. Due to the lack of a purely automated mechanism to disambiguate between site-specific sub-domains (*e.g.,* player.bbc.com) or country-specific domains (*e.g.,* bbc.co.uk), we leverage the Mozilla Public Suffix list [51] and tldextract [36] for this task. From the Alexa Top-200k websites, we collect 11,287,230 URLs which consist of 6,806,494 unique external resources that correspond to 68,828 and 196,940, respectively, unique second level domains of third- and first-parties.

Constructing the dependencies between objects in a webpage is a non-trivial task. In cases where third-party JavaScript program gets loaded into a first-party context, and then makes an AJAX request, the HTTP(S) request appears to be from the first-party (i.e. the *referrer* will be the first-party). To overcome such cases and to preserve the information on relations between the nested resource dependencies, we allow the crawler to include the URL of the third-party from which the JavaScript program was loaded by first-party.

2.1.2 Data Validation. As our main dataset relies on a single snapshot, we want to evaluate the stability of the resources loaded by websites to ensure that a single snapshot does not miss significant complexity within the ecosystem. Thus, we repeat the methodology from Section 2.1.1 on a daily basis to study how the dependency chains evolve. Unfortunately, performing daily crawls for the Alexa top-200k websites was not possible due to scalability reasons. We therefore selected 1,500 domains as a seed for the crawler. This list consists of the Alexa top-1K alongside 250 domains randomly selected from the Alexa rank ranging from 1K to 50K, and a final 250 domains randomly chosen from websites within the Alexa rank 50K–200K. This offers a broad sampling of the Alexa sites covered. In total, on a daily basis from September 15 to October 2 2018, we have collected on average 225,035 unique URLs per daily snapshot which covers 5,423 unique second level domains from the 1,500 first-parties.

Figure 2 presents the day-to-day stability of the domains we see within each website.[4] We observe that 95.07% of second level domains remain consistent across consecutive days, and only an average of 4.93% domains are absent in any two consecutive snapshots. On average, only 35 (0.66%) and 232 (4.27%) domains are absent at explicit and implicit dependency levels, respectively. Hence, we take this as a strong indicator that utilising a single snapshot is sufficient for gaining vantage into the use of third-parties.

[4]We define the (normalized) stability as the count of domains present in the dependency trees crawled on day n and also present on day $n + 1$. More specifically, let C denoting the crawled data then, stability = $\frac{C_n \cap C_{n+1}}{C_n \cup C_{n+1}}$.

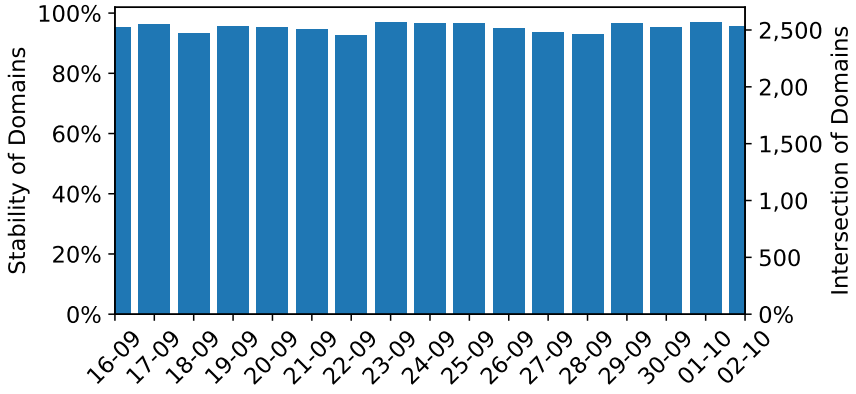


Fig. 2. Stability of day-by-day dependency trees analyzed per domain.

2.2 Meta-data collection From VirusTotal

The next challenge is to classify domains as suspicious *vs.* innocuous. For this we use VirusTotal — an online solution which aggregates the scanning capabilities provided by 68 Anti-Virus (AV) tools, scanning engines and datasets. It has been commonly used in the academic literature to detect malicious apps, executables, software and domains [21, 25, 26, 28, 33, 34, 59]. For each domain, we use the VirusTotal report API to obtain the VTscore for each third-party domain. This VTscore represents the number of AV tools that flagged the website as malicious (max. 68). The reports also contain meta-information such as the first scan date, scan history, domain name resolution (DNS) history, website or domain category, reverse DNS, and whois information. We further supplement each domain with their WebSense [58] category[5] provided by the VirusTotal's *record* API. During the augmentation, we eliminate repeating, unresponsive or invalid URLs in each dependency chain. Thus, we collect the above metadata for each second level domain in our dataset. This results in a final sample of 196,940 first-party websites, and 68,828 third-party domains.

3 EXPLORING THE CHAINS

We begin by exploring the presence and usage of implicit trust chains. We first confirm if websites do, indeed, rely on implicit trust and then explore how these chains are used. To this end, at each level of the dependency chain, we choose two metrics: number of requests and number of third-parties. The first metric, the number of requests, shows the significance or volume of resources imported from different levels, whereas the second metric characterises coverage of third-parties in different levels.

3.1 Do websites rely on implicit trust?

Overall, the Top-200k dataset makes 11,287,230 calls to 6,806,494 unique external resources, with a median of 27 external resources per first-party website. To dissect this, Table 1 presents the percentage of webpages in each Alexa range that load explicitly and implicitly trusted third-parties. Confirming prior studies [11, 38], it shows that 95% of websites import external resources, with 91% importing externally hosted JavaScript codes. More important is that around 50% of the websites *do* rely on implicit trust chains, *i.e.,* they do include third-parties to load further third-parties on their behalf. The propensity to form dependency chains is marginally higher in more popular websites; for example, 55% in the Alexa top 10K have dependency chains compared to 48% in the bottom 10K

[5]For details on the websites or domains classification, we refer the reader to WebSense's, also known as ForcePoint, domains classification repository [12].

	Alexa Rank					
	1-200K	1-10K	190-200K	10-50K	50-100K	100-200K
First-parties that trust at least one third-party which loads:						
Any Resources:						
Explicitly (Lvl. 1)	95%	95%	95%	94%	95%	95%
Implicitly (Lvl. ≥ 2)	49.7%	55.1%	47.9%	51.8%	50.23%	48%
JavaScript Resources:						
Explicitly	91%	92%	91%	91%	91%	90%
Implicitly	49.5%	55%	47.8%	51.69%	50%	47.8%

Table 1. Overview of the dataset for different ranges of the Alexa ranking. The rows indicate the proportion of Alexa's Top-X websites (with rank values lie in the rank-range such as 1-10K and 1-200K) that explicitly and implicitly trust at least one third-party (*i*) resource (of any type); and (*ii*) JavaScript code. It shows that 95% of websites import external resources, with 91% importing externally hosted JavaScript codes. Moreover, around 50% of the websites *do* rely on implicit trust chains, *i.e.,* they allow third-parties to load further third-parties on their behalf.

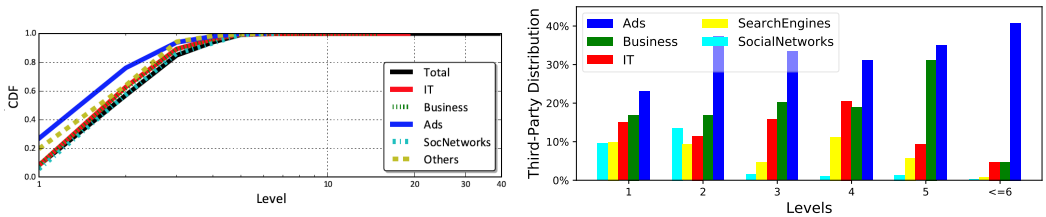


Fig. 3. **(a) CDF of dependency chain lengths (broken down into categories of first-party websites); and (b) distribution of third-party websites across various categories and levels.**

(*i.e.,* rank 190-200K). In other words, more popular websites tend to rely more on implicitly trusted third-parties.

These implicitly trusted third-parties appear at various positions in the dependency chain. Intuitively, long chains are undesirable as they typically have a deleterious impact on page load times [56] and increase attack surface. Figure 3a presents the CDF of chain length for all first-party websites. For context, websites are separated into their sub-categories.[6] It shows that 80% of the first-party websites create chains of trust of length 3 or below. However, there is also a small minority that dramatically exceed this chain length: we find that all website categories import ≈2% of their external resources from level 3 and above. In the most extreme case, we see `rg.ru` (news) with a chain containing 38 levels, consisting of mutual calls between `adriver.ru` (ad provider) and `admelon.ru` (IT website). Other notable examples include `thecrimson.com` (Harvard's student newspaper), `argumenti.ru` (news), `mundomax.com` (IT news), `lifestyle.bg` (entertainment), which have a maximum dependency level of 15. We argue that these complex configurations make it extremely difficult to reliably audit such websites, as a first-party cannot be assured of which objects are later loaded.

[6]We only include the most popular categories.

Level	Total	# Unique Resource Calls	# Unique Third-Parties	Image	JavaScript Codes	Data	Font/CSS	Uncateg.
1	9,212,245	8,866,074	57,375 (83.36%)	34.4%	30.6%	16.0%	7.8%	11.3%
2	1,566,841	1,295,322	8,617 (12.52%)	48.8%	16.7%	11.7%	3.3%	19.4%
3	405,390	223,080	1,618 (2.35%)	45.0%	12.3%	11.1%	1.3%	30.2%
4	78,107	90,984	647 (0.94 %)	41.8%	18.4%	8.0%	8.1%	23.6%
5	14,413	8,928	310 (0.45%)	40.6%	18.0%	12.8%	2.0%	26.4%
≥6	10,208	4,764	548 (0.8%)	36.6%	12.3%	13.0%	1.2%	36.8%

Table 2. Breakdown of resource types requested by the Top-200K websites across each level in the dependency chain. Total column refers to the number of resource calls made at each level.

Briefly, we also note that Figure 3a reveals subtle differences *between* different categories of third-party domains. For example, those classified as adverts are most likely to be loaded at level 1; this is perhaps to be expected, as many ad brokers naturally serve and manage their own content. In contrast, Social Network plugins and widgets (*e.g.,* Facebook plugins) are least likely to be loaded at level 1. We found that social networks are typically (99% of the times) loaded via CDNs (*e.g.,* `akamaihd.net` which is hosting the JavaScript codes belonging to Facebook) and in some cases (1% of the times) via third-parties, *i.e.,* analytics services (*e.g.,* `addthis.com`). Business third-parties are also very common: As per the Websense [12, 58] categorisation, this includes websites devoted to business firms, business associations, industry groups, *e.g.,* banks, credit unions, credit cards, and insurance. This also includes websites that provide access to business-oriented web applications and allow storage of sensitive data. Whilst the "IT" category includes websites providing information about computers, software, the Internet and related business firms, including sites supporting the sale of hardware, software, peripherals and services.

3.2 What objects exist in the chain?

The previous section has confirmed that a notable fraction of websites create dependency chains with (up to) tens of levels. We next inspect the types of resources imported within these dependency chains. We classify resources into four main types: Image, JavaScript codes, Data (consisting of HTML, JSON, XML, plain text files), and CSS/Fonts.[7] Overall, first-party websites import a median of 9 JavaScript codes and/or 6 images from external websites. Table 2 presents the volume of each resource type imported at each level in the trust chain. We observe that the make-up of resources varies dramatically based on the level in the dependency chain. For example, the fraction of images imported tends to increase with each level— this is largely because third-parties are in-turn loading images (*e.g.,* for adverts). In contrast, the fraction of JavaScript programs decreases as the level in the dependency chain increases: 30.6% of resources at level 1 are JavaScript codes compared to just 12.3% at level 3. This trend is caused by the fact that new levels are typically created by JavaScript code execution (thus the fraction of JavaScript codes is likely to deplete along the chain). However, it remains at a level that should be of concern to web engineers as this confirms a significant fraction of JavaScript code is loaded from potentially unknown implicitly trusted domains (see Section 7.1 for further discussion).

To build on this, we also inspect the *categories* of third-party domains hosting these resources. Figure 3b presents the make-up of third-party categories at each level in the chain. It is clear that, across all levels, advertisement domains make up the bulk of third-parties. We also notice

[7]We classify using the HTTP headers and URL extensions (*i.e.,*`*.js, *.html, *.css`); this allowed us to classify 85% of resources.

other highly demanded third-party categories such as search engines, Business and IT. These are led by well known providers, *e.g.,* `google-analytics.com` (web-analytics[8]) is on 68.3% of pages. The figure also reveals that the distributions of categories vary across each dependency level. For example, 23.1% of all loaded resources at level 1 come from advertisement domains, 37.3% at level 2, and 46.2% at level 3. In other words, the proportion increases across dependency levels. In contrast, social network third-parties (*e.g.,* Facebook) are mostly presented at level 1 (9.58%) and 2 (13.57%) with a significant drop at level 3. The dominance of advertisements is not, however, caused by a plethora of ad domains: there are far fewer ad domains than business or IT (see Table 3). Instead, it is driven by the large number of requests to advertisements: Even though ad domains only make-up 1.5% of third-parties, they generate 25% of resource requests. Importantly, these popular providers can trigger further dependencies; for example, `doubleclick.com` imports 16% of its resources from further implicitly trusted third-party websites. This makes such domains an ideal propagator of malicious resources for any other domains having implicit trust in it [32, 38, 39, 53, 59].

4 FINDING SUSPICIOUS CHAINS

The previous section has shown that the creation of dependency chains is widespread, and there is extensive implicit trust within the web ecosystem. This, however, does not shed light on the activity of resources within the dependency chains, nor does it mean that the implicit trust is abused by third-parties. Thus, we next study the existence of *suspicious* third-parties, which could lead to abuse of the implicit trust. Within this section we use the term *suspicious* (to be more generic than malicious) because VirusTotal covers activities ranging from low-risk (*e.g.,* sharing private data over unencrypted channels) to high-risk (malware).

				VTscore ≥ 3		**VTscore ≥ 10**		**VTscore ≥ 20**		**VTscore ≥ 40**		**VTscore ≥ 55**	
Cat	**Third-Parties**	**Total Calls**	**Suspicious JS**	Num.	Vol.	Num.	Vol.	Num.	Vol.	Num.	Vol.	Num.	Vol.
All	68,828	11,287,204	270,758 (2.4%)	1.6%	6.4%	1.2%	6.2%	1.0%	6.1%	0.6%	5.7%	≤ 0.1%	≤ 0.1%
Business	6,786	1,924,591	184,360 (9.6%)	1.5%	21.5%	1.1%	21.5%	1.0%	21.4%	0.5%	20.6%	0%	0%
Ads	1,017	2,870,482	7,924 (0.3%)	3.5%	0.1%	3.3%	0.1%	2.9%	0.1%	1.6%	≤ 0.1%	0%	0%
IT	8,619	1,646,287	10,547 (0.6%)	2.2%	3.8%	1.5%	3.6%	1.2%	3.5%	0.6%	3.0%	≤ 0.1%	≤ 0.1%
Other	52,406	4,845,844	67,927 (1.4%)	1.4%	4.6%	1.1	4.3%	0.9%	4.2%	0.6%	3.8%	≤ 0.1%	≤ 0.1%

Table 3. Overview of suspicious third-parties in each category. **Col.2-4:** number of third-party websites in different categories, the number of resource calls to resources, and the proportion of calls to suspicious JavaScript codes. **Col.5-9:** Fraction of third-party domains classified as suspicious (*Num.*), and fraction of resource calls classified as suspicious (*Vol.*), across various VTscores (i.e., ≥ 3 and ≥ 55).

4.1 Do chains contain suspicious parties?

First, we inspect the fraction of third-party domains that trigger a warning by VirusTotal. From our third-party domains, 2.5% have a VTscore of 1 or above, *i.e.,* at least one virus checker classifies the domain as suspicious. If one treats the VTscore as a ground truth, this confirms that popular websites *do* load content from suspicious third-parties via their chains of trust. However, we are reticent to rely on VTscore ≥ 1, as this indicates the remaining 67 virus checkers did not flag the domain.[9] Thus, we start by inspecting the presence of suspicious third-parties using a range of thresholds.

[8]Grouped as in business category as per VirusTotal reports.
[9]Diversity is likely caused by the virus databases used by the different virus checkers [6]

Table 3 shows the fraction of third-parties that are classified as suspicious using several VTscore thresholds. For context, we separate third-parties into their respective categories (using WebSense). The table confirms that a noticeable subset of suspicious third-party domains exist; for example, if we classify any resource with a VTscore $\geq$ 10 as suspicious, we find that 1.2% of third-party domains are classified as suspicious with 6.2% of all resource calls in our dataset going to these third-parties. Notably this only drops marginally (to 5.7%) with a *very* conservative VTscore of $\geq$ 40. We observe similar results when considering thresholds in the [3..50] range. This confirms, with a high certainty, that approximately 6% of resource calls in the dependency chains are towards domains that engage in suspicious activity (see Section 5 for further details). We will conservatively refer to domains with a VTscore $\geq$ 10 as suspicious in the rest of this analysis.

The proportion of suspicious third parties and resource calls can be related to a *prominence* metric defined in [9] that measures the frequency with which a user browsing encounters the third-party. The paper showed that the top 5 third-parties (`doubleclick.net`, `google-analytics.com`, `gstatic.com`, `google.com`, and `facebook.com`) have a prominence of 5.7 on average. The exact relationship between the prominence and the number of suspicious third-parties (and their volume of resource calls) is not important to us. However, a high prominence of a suspicious third party means that users have a high probability of becoming a target, i.e., the effect of the 1.2% suspicious third parties becomes more devastating when a user is accessing those websites multiple times. For instance, in Table 5 we show that `google-analytics.com` is the top most suspicious third-party which has a prominence of 6.20 implying that a user is hit 6.2 times by this website.

Additionally, we inspect first-party domains that inherit suspicious JavaScript resources from the explicit and various implicit levels. We focus (cf. Section 5) on JavaScript programs as active web content that poses great threats with significant attack surfaces consisting of vulnerabilities related to client-side JavaScript codes, such as cross-site scripting (XSS) and advanced phishing [38]. Table 4 shows the top first-party domains, ranked according to the number of unique suspicious third-parties in their chain of dependency. We note that the top ranked (most vulnerable) first-party domains belong to various categories such as Content Sharing, News, or IT. This indicates that there is not any single category of domains that inherits suspicious JavaScript codes. However, we note that first party websites categorised as "Business" represent the majority of most exposed domains at Level $\geq$2: 16% of first-party domains implicitly trusting suspicious JavaScript codes belonging to the Business Category. The distant second is the "News & Media" Category, and the third is "Adult". The number of suspicious JavaScript codes loaded by these first-party domains ranges from 4 to 31. For instance, we note the extreme case of `amateur-fc2.com`, which *implicitly* imports 31 unique suspicious JavaScript programs from 4 unique suspicious domains.

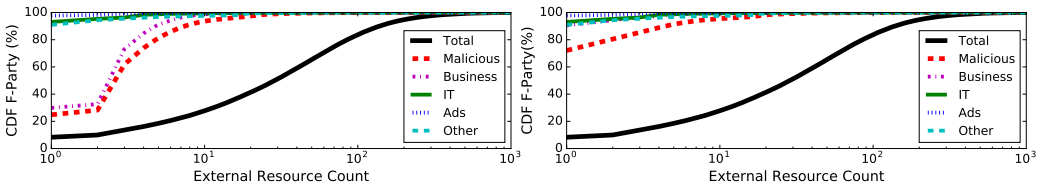
4.2 How widespread are suspicious parties?

We next inspect how widespread these suspicious third-parties are at each position in the dependency chain, by inspecting how many websites utilize them. Figure 4a displays the cumulative distribution (CDF) of resource calls to third-parties made by each first-party webpage in our dataset. Within the figure, we decompose the third-party resources into various groups (including total *vs.* suspicious). As mentioned earlier, we take a conservative approach and consider a resource suspicious if it receives a VTscore $\geq$ 10. We purposefully select a relatively low VTscore threshold to balance the need for broad coverage against high confidence[10]. Figure 4a reveals that suspicious parties within the dependency chains are commonplace: 24.8% of all first-party webpages contain at least 3 third-parties classified as suspicious in their dependency chain. Remarkably, 73% of first-party websites load resources from third-parties at least once. Hence, even though only 1.6% of

[10]Note that the difference between 3 and 10 only results in an increase of 0.2% resource calls classified as malicious.

#	First-party Domains	Alexa Rank	# Malicious JSes	Unique Susp. Domains	Category	Chain Length
Unique Suspicious Domains at Level = 1						
1	theinscribermag.com	46,242	6	5	Blogs	5
2	skynet-system.com.ua	192,549	6	5	Business	4
3	nodwick.com	194,823	13	4	Entertainment	4
4	iphones.ru	12,045	4	4	IT	4
5	privet-rostov.ru	193,024	6	4	LifeStyle	4
Unique Suspicious Domains at Level ≥ 2						
1	traffic2bitcoin.com	33,513	6	5	Games	7
2	radionetplus.ru	166,003	8	4	SW Download	6
3	studiofow.tumblr.com	85,483	11	4	Adult	4
4	amateur-fc2.com	52,556	31	4	Adult	5
5	fasttorrent.ru	24,250	9	4	File Sharing	7

Table 4. Top 5 most exposed first-party domains (with VTscore ≥ 10) ranked by the number of unique suspicious domains.



(a) All websites (b) Without google-analytics

Fig. 4. CDF of resources loaded per-website from various categories of third-parties.

third-party domains are classified as suspicious, their reach covers nearly three quarters of websites (indirectly via implicit trust).

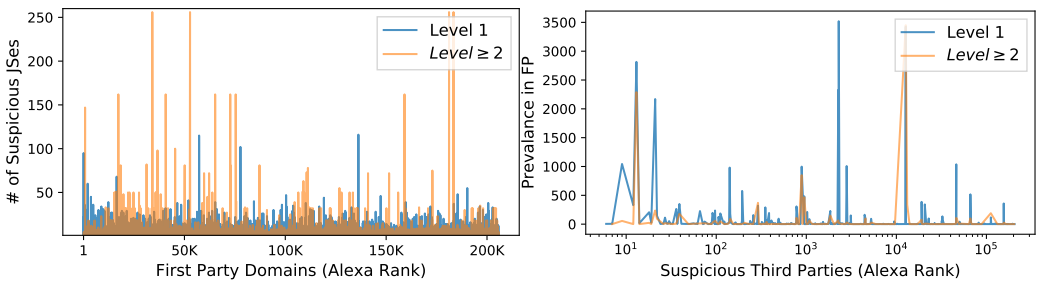
The above is demonstrated in Table 5, which presents the top 10 most frequently encountered suspicious third-party domains that are providing suspicious JavaScript codes. It can be seen that popular third-party domains exist across *many* first-party sites. The top 20% of third-party domains cover 86% (9,650,582) of all resource calls. Closer inspection shows that it is driven by one prominent third-party: `google-analytics.com`. At first, we thought that this was an error, however, during the measurement period `google-analytics.com` obtained a VTscore of 51, suggesting a high degree of certainty. This was actually caused by `google-analytics.com` loading another third-party, `sf-helper.net`, which is known to distribute adwares and spywares. It is unclear why Google was performing this. We therefore repeated these checks in October 2018, to confirm that this activity has ceased, and `sf-helper.net` is no longer loaded. To understand the impact its new de-classification has, Figure 4b shows the distribution of resource calls to third-party categories when `google-analytics.com` is benign. This reduces the number of first-party websites exposed to suspicious resources by 63%. This highlights effectively the impact of high centrality third-parties being permitted to load further resources: the infection of just one can immediately effect a significant fraction of websites.

#	Third-party Domain	Alexa Rank	# First Parties	Category
Prevalence of Third-parties at Level = 1				
1	google-analytics.com	13,200	43,156	Business (Web Analytics)
2	gravater.com	2,292	3,520	IT
3	charter.com	12,714	3,425	Business
4	vk.com	13	2,815	Social Network
5	statcounter.com	2,265	2,327	Business (Web Analytics)
Prevalence of Third-parties at Level ≥ 2				
1	charter.com	12,714	3,452	Business
2	vk.com	13	2,290	Social Network
3	livechatinc.com	888	851	Web Chat
4	onesignal.com	950	467	Business
5	rambler.ru	291	370	SearchEngine

Table 5. Top 5 most prevalent suspicious third-party domains (with VTscore ≥ 10) on level 1 (explicit trust) and beyond (implicit trust) providing resources to first-parties. Here, First-party domains having the corresponding suspicious third-party domain in their chain of dependency.

4.3 How popular are suspicious third-parties?

We next test if widespread suspicious third-parties are also highly ranked within Alexa. We treat this as a proxy for global popularity. Beyond google-analytics.com we find several other suspicious third-party domains from the Top 100 Alexa ranking. For-instance, vk.com, a social network website mostly geared toward East European countries has been used by 3,094 first-parties and is ranked 13 by Alexa. This website is found to be one of the most prevalent suspicious third-party domains at both level 1 and levels ≥ 2. An obvious reason for this domain's presence is because of other infected (malware-based) apps that try to authenticate users from such domains [46]. Other websites such as statcounter.com or gravater.com are also among the most prevalent third party domains in level 1. These websites were reported to contain malware in their JavaScript codes [10]. For instance, users in statcounter forums reported it as malicious because a JavaScript code running on its website redirects users to a malware website gocloudly.com, and forces users to click the button [13].



(a) (b)

Fig. 5. Figure (a) depicts the number of suspicious JavaScript content imported (explicitly and implicitly) by first-party domains shown according to their Alexa ranking; and (b) shows the number of impacted first-party domains as function of the ranking of domains of suspicious JavaScript codes.

More generally, we observe the presence of a wide range of Alexa ranks in the list of most prevalent domains at levels ≥ 2. In Figure 5a, we show the number of suspicious JavaScript codes imported by the first-party domains (Y-axis) according to their Alexa rank (X-axis). Overall, first-party domains import a larger number of suspicious third-party JavaScript codes at levels ≥ 2. However, the first-party domains seem to be equally vulnerable to the implicit importing of suspicious content regardless of their rank. There are exceptions though, signified by the peaks in the number of suspicious JavaScript codes — these are near exclusively driven by a large number of ≥level-2 scripts (implicit trust). We also encounter an interesting case, which we exclude from the graphs for readability purposes: The first-party domain `kikar.co.il` imports 2,592 JavaScript codes originating from the third-party `hwcdn.net`, a well-known browser hijacker that has been reported to force users to visit spam pages [54]. The VirusTotal API indicates a VTscore of 22 for this suspicious domain. We also note that 35 other first-party domains have this domain in their chain of dependency. Again, this highlights the risk of implicit trust.

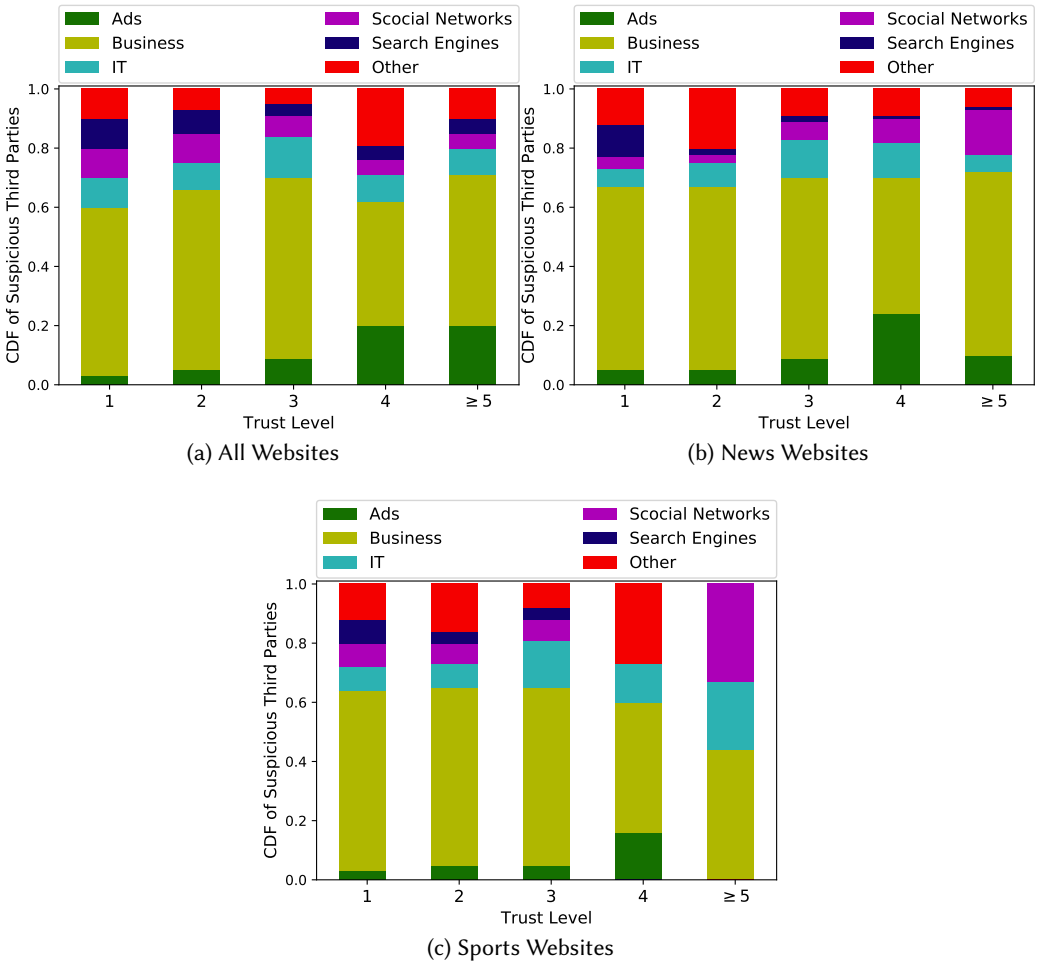
In Figure 5b we show the number of impacted first-party domains as a function of the Alexa Rank of suspicious third-party domains (limited to a maximum Alexa Rank of 1 million) — note the log scale of X-axis. Some very prevalent third-parties have a high Alexa ranking (even excluding `google-analytics.com`). For instance, note a spike around the 2000 rank, which reaches a prevalence of 3500 first-party domains at level 1. This spike is caused by `gravatar.com`, propagating suspicious JavaScript resources. This supports our statements earlier (from Table 5) where `gravatar.com` is ranked the second top most suspicious domain. Similarly, a spike around 10K rank indicates the presence of `charter.com` both at level 1 and 2 respectively. These findings demonstrate the wide variety of third-party suspicious JavaScript content loaded from various, not necessarily "obscure", third-party domains.

4.4 At which level do suspicious third-parties occur?

Next, we inspect the location(s) in the dependency chain where these suspicious third-parties are situated. This is vital, as implicitly trusted (≥level 2) resources are far more difficult for a first-party administrator to remove. Table 6 presents the proportion of websites that import at least one resource with a VTscore ≥ 10. We separate resources into their level in the dependency chain. The majority of resources classified as suspicious are requested at level 1 in the dependency chain (*i.e.,* they are explicitly trusted by the first-party). 73% of websites containing suspicious third-parties are "infected" via level 1. This might include websites that purposefully utilise such third-parties [22]. Perhaps more important, the above leaves a significant minority of suspicious resources imported via *implicit* trust (*i.e.,* level ≥ 2). In these cases, the first-party is potentially unaware of their presence. The most vulnerable category is news: over 15% of news sites import *implicitly* trusted resources from level 2 with a VTscore ≥ 10. Notably, among the 56 news websites importing suspicious JavaScript resources from trust level 3 and deeper, we find 52 loading advertisements from `adadvisor.net`. One possible reason is that ad-networks could be infected or victimized with malware to perform malvertising [40, 53].

Similar, albeit less extreme, observations can be made across Sports, Entertainment, and Forum websites. Briefly, Figure 6 displays the categories of (suspicious) third-parties loaded at each level in the dependency chain — it can be seen that the majority are classified as business. This is, again, because of several major providers classified as suspicious such as `convexity.net` and `charter.com`. Furthermore, it can be seen that the fraction of advertisement resources also increases with the number of levels due to the loading of further resources (*e.g.,* images).

Next, we again focus on JavaScript content as, when loaded, it can represent significant security risks: Our analysis is motivated by well known attack vectors underpinned by JavaScript codes, *e.g.,* malvertising [40], malware injection and exploit kits redirection. These are exemplifed by the recent



(a) All Websites (b) News Websites

(c) Sports Websites

Fig. 6. Distribution of suspicious third-party websites per category at each level, for all top-200K websites (Figure 6a) and most vulnerable first-party categories (Figures 6b, 6c).

Lv.	All		News		Sports		Entertainment		Forums	
	All	JS	All	JS	All	JS	All	JS	All	JS
1	61.30%	57.70%	75.40%	73.50%	75.70%	73.20%	69.30%	65.60%	67.40%	65.50%
2	5.20%	2.20%	13.40%	5.60%	11.10%	3.70%	8.60%	4.10%	9.10%	4.05%
3	1.30%	0.18%	2.90%	0.45%	3.60%	0.28%	2.70%	0.30%	3.20%	0.15%
4	0.22%	$\leq 0.1\%$	0.64%	0.08%	0.80%	$\leq 0.1\%$	0.70%	0.08%	0.60%	0.00%
≥ 5	$\leq 0.1\%$	0	0.002	$\leq 0.1\%$	0.001%	$\leq 0.1\%$	0.002%	$\leq 0.1\%$	$\leq$0.001%	0.00%

Table 6. Proportion of top-200K websites importing resources classified as suspicious (with VTscore $\geq$ 10) at each level.

reporting that Equifax and TransUnion were hit by a third-party web analytics script [37] [48]. Figure 7 presents the breakdown of the domain categories specifically for suspicious JavaScript resources. Clear trends can be seen, with IT (*e.g.,* `dynaquestpc.com`), Business (*e.g.,*`vindale.com`),

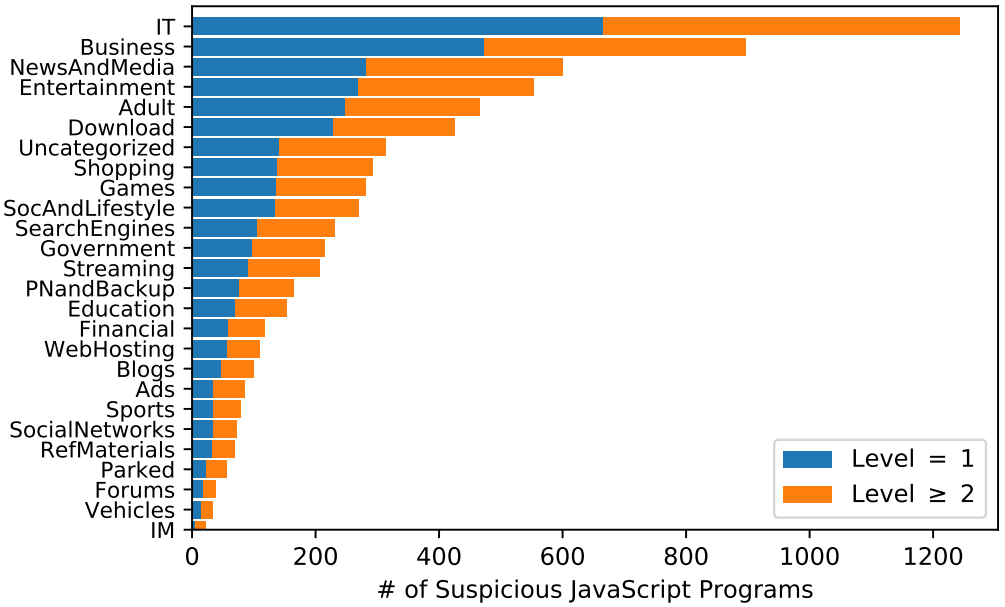


Fig. 7. Breakdown of unique, suspicious JavaScript resources at explicit and implicit trust levels. Previous work [7] used the domain category to group suspicious JavaScript resources. In the same spirit, we use the domain category to group JavaScript resources into different groups such as IT, Business, etc. Here, the Uncategorized category includes suspicious JavaScript resources whose domain's categories are *unknown* to WebSense, *e.g.*,`newmyvideolink.xyz` and `cooster.ru`. We observe that the suspicious JavaScript resources hosted by domains of IT, Business, News and Media, and Entertainment dominate at explicit and implicit trust levels.

News and Media (e.g., `therealnews.com`), and Entertainment (*e.g.*, `youwatchfilm.net`) dominating. Clearly, suspicious JavaScript resources cover a broad spectrum of activities. We observe that 70% and 67%, respectively, of Business (Web analytics) and Ads JavaScript codes are loaded from level $\geq$ 2 in contrast to 17% and 31% of JavaScript programs of Government and Shopping loaded at level 1.

We next strive to quantify the level of suspicion raised by each of these JavaScript programs. Intuitively, those with higher VTscores represent a higher threat as defined by the 68 AV tools used by VirusTotal. Hence, Figure 8 presents the cumulative distribution of the VTscores for all JavaScript resources loaded with VTscore > 0. We separate the JavaScript programs into their location in the dependency chain. Clear difference can be observed, with level 2 obtaining the highest VTscore (median 52). In fact, 78% of the suspicious JavaScript resources loaded on trust level 2 have a VTscore > 52 (indicating *very* high confidence).

This is a critical observation; whereas suspicious third-parties at level 1 can be ultimately removed by first-party website operators if flagged as suspicious, this is much more difficult for implicitly trusted resources further along the dependency chain. If the intermediate (non-suspicious) level 1 resource is vital for the webpage, it is likely that some operators would be unable or unwilling to perform this action. The lack of transparency and the inability to perform a vetting process on implicitly trusted loaded resources further complicates the issue. It is also worth noting that the VTscore for resources loaded further down the dependency chain is lower (*e.g.*, level 4). For example, 80% of level 4 resources receive a VTscore below 5. This suggests that the activity of these resources is more contentious, with a smaller number of AV tools reaching consensus. It is

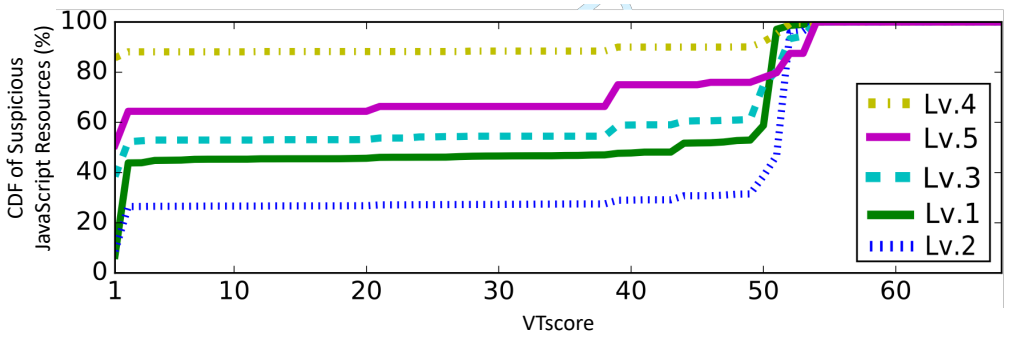


Fig. 8. CDF of VTscores for JavaScript programs (with VTscores > 0) at different levels in the chain.

impossible to state the reason for this, hence in Section 5 we analyze the dynamic activities of these JavaScript content.

5 ANALYSIS OF SUSPICIOUS JAVASCRIPT RESOURCES

JavaScript codes are arguably the most dangerous resource to import, as JavaScript codes have the potential to execute diverse functions (including the downloading of further resources). Thus, we proceed to inspect the activities of the 7,166 JavaScript resources that were classified as suspicious in our dataset. We achieve this by executing the JavaScript resources in an isolated sandbox, and studying their activities.

5.1 Methodology

We use a dedicated testbed composed of three Virtual machines (VMs) that connect to the Internet via a computer running the Cuckoo sandbox and `tcpdump`. These VMs are configured with Windows 7, and are utilised to log all system-level events and to intercept all traffic being transmitted between the virtual machines and the Internet. Moreover, we use `Volatility` [55] to collect and analyse memory dumps of JavaScript code running in the browser. `Volatility` allows us to reveal information (*i.e.,* kernel-level processes and network connections) about the analysed JavaScript codes. This allows us to observe the traffic generated by each JavaScript code when it is rendered by the browser. For instance, our logs keeps a record of the network traffic generated, all file operations, memory changes, registry changes *etc.*.

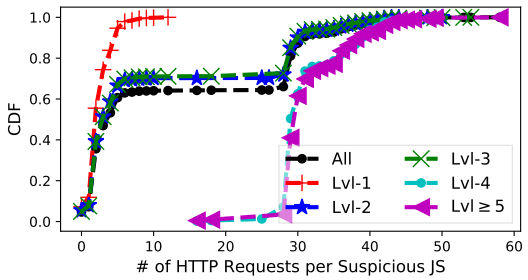
For each test, we first create an HTML document and inject suspicious JavaScript code via the `<script>` tag. We then load the HTML in the browser of our VM testbed. We configured our testbed to wait 120 seconds for each target JavaScript code, embedded in an HTML code, to be rendered by the VM browser. The yielded logs are stored in a JSON object and pushed to our storage server for further analysis. It took, on average, an additional 3 seconds transferring and saving data at our server. Prior to each test, we turn-off and restore the VM to a clean snapshot. This ensures that any malicious software downloaded by prior JavaScript code's test does not remain on the VM. We share the code and data for the testbed at `https://wot19submission.github.io`.

#	Level	JavaScript Code	Category	VTscore	A-Rank	#HTTP	#Domains	Observed Behavior
1	Lvl-1	hxxp://pinshan.com/js/union/new-play-1.js	Business	12	25,574	12	5	PUP[11] activity e.g., Installing Fake AV and mediaplayers
2	Lvl-1	hxxp://newyx.net/js/dui_lian.js	Games	11	22,057	12	6	Displaying annoying ads and perform click fraud
3	Lvl-1	hxxp://loxblog.com/fs/clocks/02.js	IT	10	86,505	10	3	Installing additional SW with elevated privileges
4	Lvl-1	hxxp://mecum.com/js/jquery.fancybox.pack.js	Business	13	51,897	9	3	PUP activity, Installing Fake AV and mediaplayers
5	Lvl-1	hxxp://bubulai.com/js/xp.js	Enter.	10	117,261	9	5	Displaying annoying ads and perform click fraud
1	Lvl≥2	hxxp://yourjavascript.com/3439241227/blog.js	PNandBackup	13	2,007,688	58	51	Displaying annoying ads and perform click fraud
2	Lvl≥2	hxxp://negimemo.net/alichina/login.js	SW Download[12]	10	13,093,855	49	21	Displaying annoying ads and perform click fraud
3	Lvl≥2	hxxp://funday24.ru/js/c/funday-index.js	News	11	2,017,900	42	9	Displaying annoying ads and perform click fraud
4	Lvl≥2	hxxp://netcheckcdn.xyz/optout/set/strtm.js	Business	14	18,064,762	42	7	Displaying annoying ads and perform click fraud
5	Lvl≥2	hxxp://pushmoneyapp.com/js/main.js	Business	17	8,757,970	41	6	Installing additional SW with elevated privileges

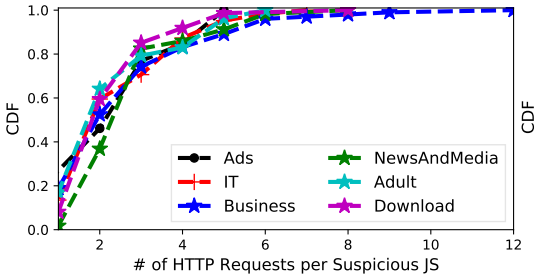
Table 7. Top 5 suspicious JavaScript resources measured by number of HTTP requests generated. We separate into JavaScript code loaded at Level 1 (upper part of table) and Level ≥2 (lower). Here 'A-Rank' and 'Category' represent *Alexa* rank and category of the domain of the JavaScript code, respectively. Here, PUP stands for Potentially Unwanted Programs, including "bogus" software such as free screen-savers or fake AV scanners that surreptitiously generate advertisements or perform redirections to collect personal identifiable information. SW Download means websites that share or facilitate downloading software executables. Note that PDF readers may render these suspicious links and expose readers of the paper to potential risks; therefore, we have replaced `http://` with `hxxp://` to avoid PDF rendering and potential risks.

5.2 Results

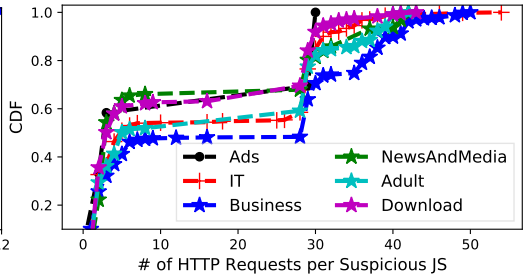
Using our sandbox testbed, we next measure and briefly discuss which resources are accessed by suspicious JavaScript programs, as well as any dropfiles that are generated on the VM.



(a) CDF of number of generated HTTP requests per suspicious JS at different dependency levels.



(b) CDF of the number of generated HTTP requests per suspicious JS at Level = 1



(c) CDF of number of generated HTTP requests per suspicious JS at Level ≥ 2

Fig. 9. CDFs of the number of HTTP requests generated per suspicious JavaScript resources viewed: (*i*) across different levels of the dependency chain and categories of domains; (*ii*) Level = 1; and (*iii*) Level ≥ 2.

5.2.1 HTTP Request Frequency. We start by inspecting the underlying HTTP requests triggered by the JavaScript programs. Table 7 provides a list of the JavaScript resources that generate the most HTTP requests (separated into implicit and explicitly trusted). There is significant network activity generated by the suspicious JavaScript resources within our testbed, with downloads initiated at various locations in the dependency chain: 44.7% of requests are triggered at level 1 (explicit trust), whereas 55.3% at level ≥ 2 (implicit trust).

To explore this further, Figure 9a presents the distribution of the number of HTTP requests generated per suspicious JavaScript. The figure splits the JavaScript programs into their respective positions in the dependency chains. Although 47% of JavaScript resources generate fewer than 5 requests, there are notable differences among the different levels. JavaScript resources at level 1 generate the fewest HTTP requests (median 2), yet level ≥ 4 are extremely active (median 30 requests). 36% of the JavaScript programs imported from level 5 generate at least 30 HTTP requests in contrast to 15% of the JavaScript programs sourced from level 2. This is in contrast to a typical behavior of legitimate JavaScript programs that have been previously measured to generate on average just 4 HTTP requests [47].

Furthermore, VirusTotal shows that those at level 1 tend to have lower VTscores (average 13), compared to those at ≥ 2, which tend to have a higher score (average 21). This is worrying as

resources loaded further down the dependency chain are the most opaque to the website operator. The most regularly observed JavaScript at level 1 is `new-play-1.js`, a relatively highly ranked (22,574 Alexa) script which downloads dropfiles. In contrast, at level ≥ 2, the most regularly observed JavaScript is `blog.js`, which show intrusive adverts that perform click fraud.

We are also interested in how behaviours might differ across categories of website. Hence, Figure 9b and 9c separate the JavaScript resources into their respective content categories. They then plot the distribution of number of requests per JavaScript within these categories. Whereas those at level 1 (explicit trust) exhibit relatively similar traits across all categories (Figure 9b), we find that those at level ≥ 2 (implicit trust) have far more divergence across categories (Figure 9c). Those classified as Business, IT or Adult are the most active, whereas News, Ads and Download generate the fewest HTTP requests. This is largely driven by the fact that most Business (*i.e.,* a subcategory of web-analytics) domains download more JavaScript codes, which then subsequently trigger further downloads (creating a cumulative effect). In contrast, other categories (*e.g.,* IT, Adult and Blogs) tend to download more static content, *e.g.,* images (which do not trigger further requests). When inspecting what exactly the requests contain, we find that the overwhelming majority are downloading dropfiles (see Section 5.2.3). A remarkable 99.5% of all suspicious JavaScript codes download at least one dropfile, with the vast majority (98.62%) involving malvertising and click fraud (as identified via VirusTotal). This creates a heavy traffic footprint: 22% of HTTP requests are downloading dropfiles (further discussed in Section 5.2.3).

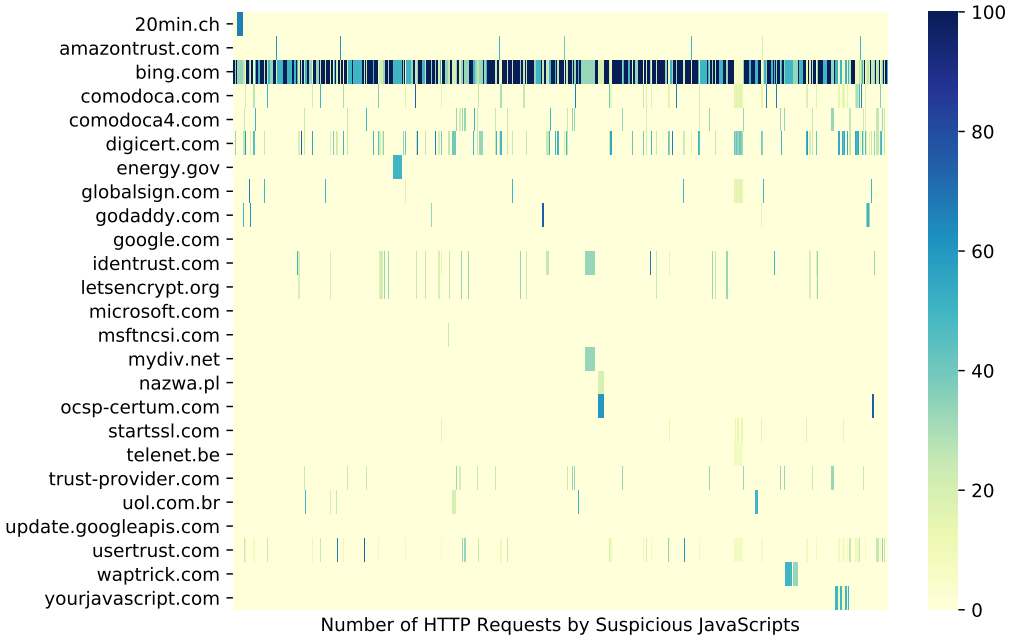


Fig. 10. Heatmap of number of requests to domains by suspicious JavaScript codes and histogram of top 25 contacted domains by suspicious JavaScript codes.

5.2.2 HTTP Request Targets. We next inspect the domains that these requests are accessing, *i.e.,* the domains hosting the content and files downloaded. For ease of presentation, we consider the top 25 domains targeted by the suspicious JavaScript codes (in terms of total number of HTTP requests targeting them). Figure 10 presents a heatmap illustrating the number of requests to them, with the

X-axis showing the suspicious JavaScript code and the Y-axis listing the targeted domains. The heat is defined as the fraction of requests that each JavaScript issued to each domain. We find that most JavaScript codes have a distinct preference towards a small set of domains. For instance, 18% of JavaScript programs submit over 50% of their HTTP requests to a single domain. One particularly popular domain is `bing.com`; 65% of all suspicious JavaScript programs access this domain at least once. Closer inspection suggests that most JavaScript resources targeting `bing.com` undertake some form of search engine optimisation (SEO), *e.g.*, launching exploits to elevate the ranking of certain URLs in the results [20, 31].

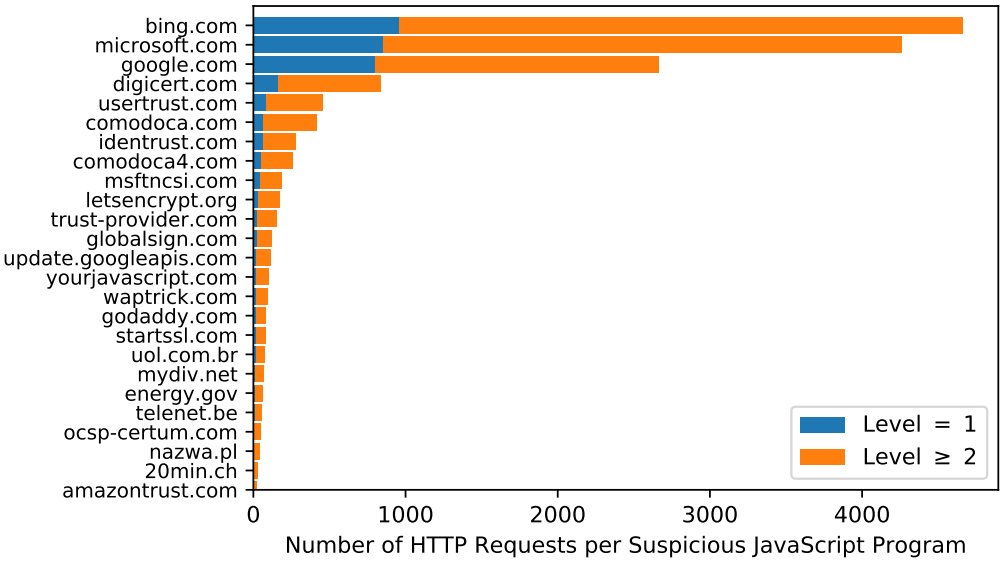
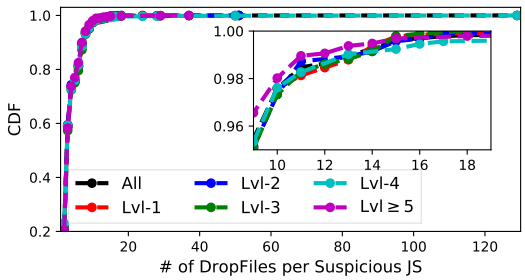


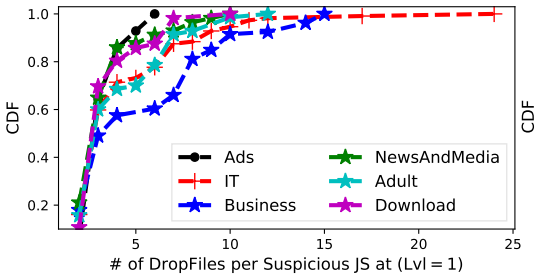
Fig. 11. Number of HTTP fetch requests issued by suspicious JavaScript resources, at level = 1 vs. level ≥ 2, to top 25 domain. Here X-axis shows the sum over all the loads of all the JavaScript programs across all analysed domains. The figure shows that most commonly occurring HTTP fetch is to `bing.com`.

Figure 11 also presents the count of HTTP requests across the top 25 targeted domains. We separate JavaScript codes into level 1 *vs.* level ≥ 2. We observe that the majority of fetches are triggered by level 1 (*i.e.,* they are explicitly trusted by the first party). However, we also note a large number of fetches to these domains are from JavaScript resources loaded at level 2. Revisiting our earlier example, 79.5% of HTTP requests to `bing.com` are triggered by level ≥ 2, indicating that the first party domain might be unaware that they are responsible for this attack (these requests are primarily for search engine manipulation [31]). As well as search engines, we note the existence of various certification authorities. We leave further inspection of these activities to future work, but conjecture that one of the reasons behind sending requests to the certification authorities is that digitally signed suspicious JavaScript resources can bypass system protection mechanisms that only install or launch programs with valid signatures. Such malware could also evade anti-virus programs which often forego scanning signed binaries.

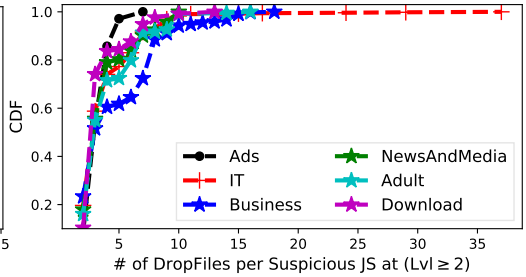
5.2.3 Analysis of Dropfiles. The above has revealed that a large number of suspicious JavaScript resources download files. The use of these dropfiles is commonplace, and they are often used during the infection process [3]. For instance, these files can potentially contain the unpacked malware binary that could potentially present worrisome vulnerabilities. Hence, we next inspect the creation of local files by JavaScript.



(a) Dropfiles by JavaScript Resources



(b) Dropfiles by JavaScript resources at Level = 1



(c) Dropfiles by JavaScript resources at Level ≥ 2

Fig. 12. CDF of dropfiles downloaded and operated (i.e., read/write operation) by suspicious JavaScript codes.

We observe a significant number of active memory operations[13] as depicted by the number of dropped files. Note that dropfiles are executables (*e.g.,* malware, Exploitkits, Trojans, *etc.*) exploiting browsers to download and execute code without user consent. As previously identified, 99.5% of the suspicious JavaScript codes generate at least one dropfile, indicating that this is the most common activity undertaken. We observe significant differences in the number of dropfiles downloaded by each JavaScript though. Figure 12 depicts the number of files dropped by the JavaScript content as a CDF. Whereas the majority download below 5, a small minority exceed 30. 22% of JavaScript codes generate at least 8 files from memory by compiling the dynamically loaded code in active memory and saving them to OS specific executables (*i.e.,* a "Trojan") confirming that memory exploits are being used by these JavaScript codes.

Table 8 presents the top-10 JavaScript codes based on how many dropfiles they generate. Some of these JavaScript codes are extremely active. For example, `xbigg.com/adv.js` (loaded at level 1) downloads 16 files. Although there is not any significant difference between the number of dropfiles per level, most active JavaScript code (`hxxp://yourjavascript.com/3439241227/blog.js`) is loaded at level 4 and downloads 129 files. It is also interesting to observe that the resources at level ≥ 2 tend to have higher VTscores, indicating that their activities are blocked by a large number of virus checkers. The actual content of the files are quite diverse.

[13] Active memory operation mean processes that operates in memory. New types of malware differ from the traditional ones in the sense that they dynamically load suspicious codes from servers controlled by cybercriminals and run suspicious instructions from memory (*i.e.,* random access memory (RAM)).

#	Level	JavaScript Code	# of Drop files	% of Mal. DropFiles	VTscore	Mal. Type	Observed Behavior
1	Lvl-1	hxxp://xbigg.com/adv.js	16	64%	9	AdCB	Displaying annoying ads and perform click fraud
2	Lvl-1	hxxp://passback.free.fr/webmails/js/edito.js	10	82%	4	AdCB	Displaying annoying ads and perform click fraud
3	Lvl-1	hxxp://via-midgard.info/engine/ajax/loginza.js	10	89%	5	AdCB	Displaying annoying ads and perform click fraud
4	Lvl-1	hxxp://pokesnipe.de/js/app.min.js	10	90%	4	Torjan	Installing additional SW with elevated privileges
5	Lvl-1	hxxp://hdvideo18.com/includes/ace.min.js	8	100%	3	AdCB	Displaying annoying ads and perform click fraud
1	Lvl≥2	hxxp://yourjavascript.com/3439241227/blog.js	129	20%	15	AdCB	Displaying annoying ads and perform click fraud
2	Lvl≥2	hxxp://s2d6.com/js/globalpixel.js	25	69%	11	AdCB	Displaying annoying ads and perform click fraud
3	Lvl≥2	hxxp://cmsdude.org/wp-includes/js/jquery/jquery.js	12	71%	11	AdCB	Displaying annoying ads and perform click fraud
4	Lvl≥2	hxxp://widih.com/js/like.js	10	90%	10	PUP	PUP activity, Installing Fake AV and mediaplayers
5	Lvl≥2	hxxp://pichak.net/blogcod/clock/04/clock.js	8	100%	10	Exploitkit	Installing Exploitkit and performing Web redirects

Table 8. Top 5 suspicious JavaScript codes with dropfiles (*i.e.,* executables such as malware, Exploitkits, Trojans, *etc.* are exploiting the browser to download and execute code without user consent) at explicit and implicit dependency level. AdCB means Adware and Click Bots. Note that PDF readers may render these suspicious links and expose readers of the paper to potential risks; therefore, we have replaced `http://` with `hxxp://` to avoid PDF rendering and potential risks.

Figure 13 plots the distribution of file types, as classified by VirusTotal (see Section 2.2). We exclude the 8% which are encrypted, and therefore cannot be examined. The vast majority of remaining files (98.62%) are Adware and Click bots, suggesting that these types of financial gain are a major driving force in this domain. The remainder are Potentially Unwanted Programs (0.52%), Exploitkits (0.36%), Adware and Click Bots (98.62%), and Trojan (0.50%). For instance, `videowood.tv/assets/js/poph.js` uses and exploits `eval()` — JavaScript's dynamic loading method — to download and execute `1832-fc204a9bcefeab3d.exe` (with VTscore=5). This then enables the attacker to take over web browser for displaying a wide range of adverts and garner fraudulent clicks.

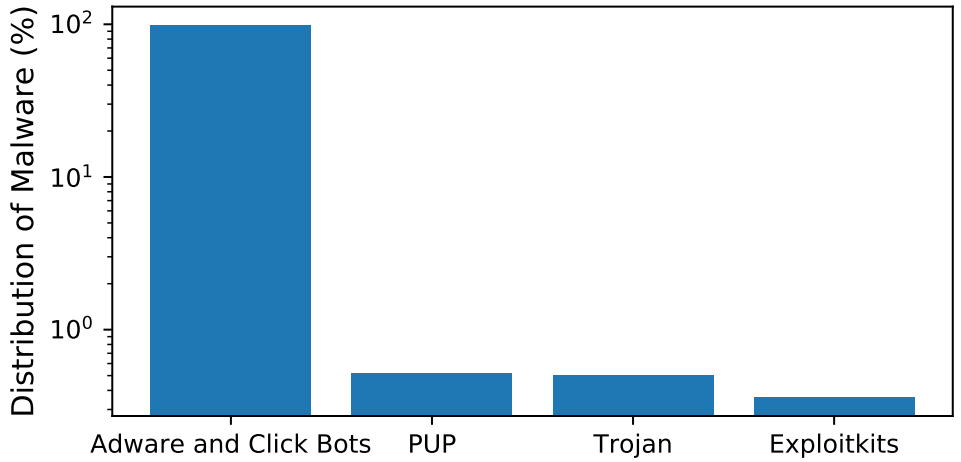


Fig. 13. Histogram of type of malware (*i.e.,* dropfiles) as per VirusTotal reports (*cf.* Section 2.2).

6 RELATED WORK

There has been a wealth of research into the utilisation and exploitation of third-parties [23, 24, 38, 44, 49, 50] and third-parties blacklists analysis [17, 18]. Falahrastegar *et al.* [11] inspected the use of third-parties across top Alexa websites, exploring how third-party operators differ based on region. Nikiforakis *et al.* [44] demonstrated in 2012 that large proportions of websites rely on JavaScript libraries hosted on ill-maintained external web servers, making JavaScript exploits trivial. Wang *et al.* [57] studied use of third parties in China, highlighting differences to Western studies. Lauinger *et al.* [38] led a further study, classifying sensitive libraries and the vulnerabilities caused by them. Gomer *et al.* [14] analysed users' exposure to tracking in the context of search, showing that 99.5% of users are tracked by popular trackers within 30 clicks. Further, Hozinger *et al.* [19] found 61 JavaScript exploits and empirically defined three main attack vectors.

Our work differs quite substantially from these in that we are not interested in the JavaScript code itself, nor the simple presence of third-party domains in a webpage. Instead, we are interested in *how* third-parties are loaded, and their use of "implicit" trust (*i.e.,* dependency chains). In contrast to our work, these prior studies ignore the presence of dependency chains and treat all third-parties as "equal", regardless of where they are loaded in the dependency chain. Closer to our own work is Bashir *et al.* [2], who studied websites' resource inclusion trees and analyzed retargeted ads using crowdsourcing. This allowed them to identify and classify ad domains, as well as predominant cookie matching partners in the ad exchange environment. Our study is far broader, and sheds light on dependency chains across many different types of websites rather than simply inspecting

advertisements. More related is Kumar *et al.* [35], who recently characterized websites' resource dependencies on third-party services. In-line with our work, they found that dependency chains are widespread. This means, for example, that 55% of websites are prevented from fully migrating to HTTPS by their dependencies. Their focus was not, however, on identifying suspicious or malicious activities. To the best of our knowledge, this paper is the first study to analyze the role of implicit trust from a security perspective to better understand the role of dependency chains in loading suspicious third-party content. Compared to our preliminary results [27], this article introduces significant additional material. Specifically, (*i*) we introduce an empirical evaluation of the temporal behaviour within our dataset (Section 2.1.2); (*ii*) we expand our analysis to include other categories of websites (Section 4.4); and (*iii*) we design a testbed to analyse the nature and behaviour of suspicious JavaScript codes (Section 5).

7 DISCUSSION AND LIMITATIONS

In this Section, we summarise our key findings and explore simple solutions that may mitigate the impact of the vulnerabilities discussed. We then proceed to highlight some of the key limitations of our work.

7.1 Discussion and Mitigation

Our measurement results have identified a number of websites that load resources via implicit trust (*i.e.,* at level > 1 in the dependency chain). We have also confirmed that these chains often contain suspicious JavaScript resources, which expose users to risks. Unfortunately, these are not necessarily trivial to identify without appropriate expertise. Hence, the presence of these dependency chains can create challenges for identifying and filtering such resources. From the perspective of the first-party website, filtering an unwanted dependency can only be done by removing an intermediate third-party in the chain. This can be problematic if the dependency is performing a critical task (which may have taken a lot of development time to integrate). Similarly, many developers may simply not be aware of this practice and might therefore not know to check the dynamic loading of such resources.

To minimise such risks, users may leverage security and privacy preserving tools such as NoScript [30] to reduce the risk of (suspicious) JavaScript execution. However, an ordinary user is not expected to be well aware of such risks, or to install security tools. Hence, there are several methods that could be used by third-party services, resource providers, websites or browser developers to minimise the adverse impact of including resources from potentially suspicious third-parties. Most obviously, web developers and site operators should be made more aware of the risks identified in this paper. This is particularly the case as these stakeholders have the capacity to curtail the problem by blocking any resources that depend on other malicious domains. The challenge here is providing greater transparency. This could, for example, be achieved by website operators running standard development tools and using VirusTotal to classify each domain contacted. There would also be value in sharing such information across websites (*e.g.,* to crowd source a blacklist of suspicious third-party resources which depend on other third parties). Of course, this list should be communicated to the third-party operators, as they may be unaware themselves of their dependency chain. We argue that such operators should also monitor their chains to ensure that they do not load any malicious resources.

Much of these checks could be automated within the browser. Existing AV tools could be used to block malware blacklists that are loaded via implicit trust. Implementation of best practices such as sub-resource integrity checks can also mitigate issues. Cross-origin resource sharing checks [42] could similarly be performed to prevent third-party resources gaining access to the first party

context (*e.g.,* to access cookies). Websites should also ensure they utilise appropriate security headers to communicate their access policies (*e.g.,* using Access-Control-Allow-Origin).

7.2 Limitations

It is also worthwhile highlighting challenges and limitations within our work. As with many other papers, our study is dependent of the VirusTotal reports (and therefore the classification by various antivirus tools in VirusTotal). This was because malicious domains may circumvent a specific antivirus (AV) tool and, as suggested by previous studies [1, 4], some AV tools may not always report reliable results. Our experiments confirm this, as we found that AV tools often gave conflicting results (*i.e.,* they did not necessarily agree on classifications). We therefore used the VTscore records as an indicator of whether a domain (third party domains in this study) is benign or suspicious. We acknowledge that VirusTotal is not perfect and therefore there may be noise within our classifications. To limit any issues, we do not rely on classifications by a single AV and, instead, set a VTscore threshold of above 10. This parameter was derived from a number of experiments and, where possible, we have presented results for multiple VTscore settings.

We also highlight the target of our study is a potentially dynamic landscape. Our reports from VirusTotal were captured in 2016, and our measurement are likely to not include domains which have been tagged as malicious in other years or periods that were not scanned by VirusTotal in 2016. Additionally, domains can be given an abnormally high VTscore for reasons identified earlier in this paper (*e.g.,* malvertizing campaigns, *etc.*). While more accurate measurements could be achieved by scanning all URLs (rather than all domains) using VirusTotal's API, this solution might not be viable for a large-scale analysis due to limitations in the allowable request rate of the VirusTotal API.

Moreover, we would also highlight that similar issues exist when categorising websites (*e.g.,* as Business, IT, Adult *etc.*). WebSense did not return proper categories for 10% of third-party websites, and 15% of resources loaded in the dependency dataset. From these uncategorised websites, we found that 10% were deemed suspicious by VirusTotal, yet limitations in our methodology prevented us from a deeper understanding of their purpose. We therefore leave these aspects to future work.

8 CONCLUDING REMARKS AND FUTURE WORK

This paper has explored dependency chains in the web ecosystem. Inspired by the lack of prior work focusing on how resources are loaded, we found that over 40% of websites *do* rely on implicit trust. Although the majority (84.91%) of websites have short chains (with levels of dependencies below 3), we found first-party websites with chains exceeding 30 levels. The most common *implicitly* trusted third-parties are well known operators (*e.g.,* `doubleclick.net`), but we also observed various less known implicit third-parties. We hypothesised that this might create notable attack surface. To confirm this, we classified the third-parties using VirusTotal to find that 1.2% of third-parties are classified as potentially malicious. Worryingly, our "confidence" in the classification actually increases for implicitly trusted resources (*i.e.,* trust level ≥ 2), where 78% of suspicious JavaScript resources have a VTscore > 52. In other words, more implicitly trusted JavaScript resources have higher VTscores than explicitly trusted ones. These resources have remarkable reach — largely driven by the presence of highly central third-parties, *e.g.,* `google-analytics.com`. With this in mind, we performed sandbox experiments on the suspicious JavaScript to understand their actions. We witnessed extensive download activities, much of which consist of downloading dropfiles and malware. It was particularly worrying to see that JavaScript resources loaded at level ≥ 2 in the dependency chain tended to have more aggressive properties, particularly as exhibited by their higher VTscore. This exposes the need to tighten the loose control over indirect resource loading and implicit trust: it creates exposure to risks such as malware distribution, search engine

optimization (SEO) poisoning, malvertising and exploit kit redirection. We argue that ameliorating this can only be achieved through transparency mechanisms that allow web developers to better understand the resources on their webpages (and the related risks).

This is only the first step in our research agenda. Most notably, we wish to perform longitudinal measurements to understand how these metrics of maliciousness evolve over time. We are particularly interested in understanding the (potentially) ephemeral nature of threats besides the inspection of temporal dynamics of resource dependency chains (see Section 2.1.2). Without this, we are reticent to draw long-term conclusions. Another line of work is understanding how level $\geq$ 1 JavaScript content creates inter-dependencies between websites. This is particularly noteworthy among hypergiants (*e.g.,* Google), who are present on a large number of first-party websites. Deep diving into other forms of vulnerabilities (*e.g.,* cascading style-sheets) would also be key for obtaining a wider understanding.

Similarly, as we do not execute JavaScript code in the context of the actual page, it would be interesting to analyze the JavaScript running in the actual context. However, we are aware of the complexity involves, *i.e.,* it becomes difficult to extract all activities related to the individual JavaScript code. As a future work it would be interesting to further investigate their behaviour. For instance, we intend to perform graph analysis to understand how removing these popular third-parties may impact the presence of interconnected suspicious third-parties. We are also keen to explore the efficacy of strategies like the same-origin policy, *e.g.,* to see how much third-party code is running in the first-party's context and thus can access its cookies. By opening our datasets and scripts to the wider research community, we hope that this will engender further research to help address the issues observed.

REFERENCES

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014.* https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket

[2] Muhammad Ahmad Bashir, Sajjad Arshad, William K. Robertson, and Christo Wilson. 2016. Tracing Information Flows Between Ad Exchanges Using Retargeted Ads. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.* 481–496. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/bashir

[3] Boldizsár Bencsáth, Gábor Pék, Levente Buttyán, and Márk Félegyházi. 2012. Duqu: Analysis, detection, and lessons learned. In *ACM European Workshop on System Security (EuroSec)*, Vol. 2012.

[4] Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). 2015. *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1.* IEEE Computer Society. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7174815

[5] Tomasz Bujlow, Valentín Carela-Español, Josep Sole-Pareta, and Pere Barlet-Ros. 2017. A Survey on Web Tracking: Mechanisms, Implications, and Defenses. *Proc. IEEE* 105, 8 (2017), 1476–1510. https://doi.org/10.1109/JPROC.2016.2637878

[6] Julio Canto, Marc Dacier, Engin Kirda, and Corrado Leita. 2008. Large scale malware collection: lessons learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems.* Citeseer.

[7] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018.* 1515–1532. https://doi.org/10.1145/3243734.3243860

[8] CHARLES DUHIGG. 2012. How Companies Learn Your Secrets. https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html?pagewanted=all.

[9] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016.* 1388–1401. https://doi.org/10.1145/2976749.2978313

[10] IBM XForce Exchange. 2005. StatCounter session hijack. https://exchange.xforce.ibmcloud.com/vulnerabilities/20506.

[11] Marjan Falahrastegar, Hamed Haddadi, Steve Uhlig, and Richard Mortier. 2014. The Rise of Panopticons: Examining Region-Specific Third-Party Web Tracking. (2014), 104–114. https://doi.org/10.1007/978-3-642-54999-1_9

[12] Forcepoint. 2019. Master Database URL Categories | Forcepoint. https://www.forcepoint.com/product/feature/master-database-url-categories.

[13] Stat Counter Forum. 2016. http://www.statcounter.com/ counter/counter.js has malware inside it ! https://forum.statcounter.com/threads/http-www-statcounter-com-counter-counter-js-has-malware-inside-it.43792/.

[14] Richard Gomer, Eduarda Mendes Rodrigues, Natasa Milic-Frayling, and Monica M. C. Schraefel. 2013. Network Analysis of Third Party Tracking: User Exposure to Tracking Cookies through Search. In *2013 IEEE/WIC/ACM International Conferences on Web Intelligence, WI 2013, Atlanta, GA, USA, November 17-20, 2013*. 549–556. https://doi.org/10.1109/WI-IAT.2013.77

[15] Google. 2018. Headless chromium. https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md.

[16] Saul Hansell. 2006. AOL Removes Search Data on Vast Group of Web Users. http://query.nytimes.com/gst/fullpage.html?res=9504e5d81e3ff93ba3575bc0a9609c8b63. *New York Times* (2006).

[17] Saad Sajid Hashmi, Muhammad Ikram, and Mohamed Ali Kaafar. 2019. A Longitudinal Analysis of Online Ad-Blocking Blacklists. *arXiv preprint arXiv:1906.00166* (2019).

[18] Saad Sajid Hashmi, Muhammad Ikram, and Stephen Smith. 2019. On Optimization of Ad-blocking Lists for Mobile Devices. In *Proceedings of MobiQuitous 2019–16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 1–8.

[19] Philipp Holzinger, Stefan Triller, Alexandre Bartel, and Eric Bodden. 2016. An In-Depth Study of More Than Ten Years of Java Exploitation. In *Proceedings of the 2016 ACM Conference on Computer and Communications Security (CCS '16)*.

[20] Fraser Howard and Onur Komili. 2010. Poisoned search results: How hackers have automated search engine poisoning attacks to distribute malware. *Sophos Technical Papers* (2010), 1–15.

[21] Damilola Ibosiola, Ignacio Castro, Gianluca Stringhini, Steve Uhlig, and Gareth Tyson. 2019. Who Watches the Watchmen: Exploring Complaints on the Web. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 729–738. https://doi.org/10.1145/3308558.3313438

[22] Damilola Ibosiola, Benjamin Steer, Alvaro Garcia-Recuero, Gianluca Stringhini, Steve Uhlig, and Gareth Tyson. 2018. Movie Pirates of the Caribbean: Exploring Illegal Streaming Cyberlockers. *International AAAI Conference on Web and Social Media (ICWSM)* (2018).

[23] Muhammad Ikram, Hassan Jameel Asghar, Mohamed Ali Kâafar, and Anirban Mahanti. 2014. On the Intrusiveness of JavaScript on the Web. In *Proceedings of the 2014 CoNEXT on Student Workshop, CoNEXT Student Workshop '14, Sydney, Australia, December 2, 2014*. 31–33. https://doi.org/10.1145/2680821.2680837

[24] Muhammad Ikram, Hassan Jameel Asghar, Mohamed Ali Kâafar, Anirban Mahanti, and Balachander Krishnamurthy. 2017. Towards Seamless Tracking-Free Web: Improved Detection of Trackers via One-class Learning. *PoPETs* 2017, 1 (2017), 79–99. https://doi.org/10.1515/popets-2017-0006

[25] Muhammad Ikram, Pierrick Beaume, and Mohamed Ali Kâafar. 2019. DaDiDroid: An Obfuscation Resilient Tool for Detecting Android Malware via Weighted Directed Call Graph Modelling. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, ICETE 2019 - Volume 2: SECRYPT, Prague, Czech Republic, July 26-28, 2019*. 211–219. https://doi.org/10.5220/0007834602110219

[26] Muhammad Ikram and Mohamed Ali Kâafar. 2017. A First Look at Mobile Ad-Blocking Apps. In *16th IEEE International Symposium on Network Computing and Applications, NCA 2017, Cambridge, MA, USA, October 30 - November 1, 2017*. 343–350. https://doi.org/10.1109/NCA.2017.8171376

[27] Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kâafar, Noha Loizon, and Roya Ensafi. 2019. The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 2851–2857. https://doi.org/10.1145/3308558.3313521

[28] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kâafar, and Vern Paxson. 2016. An Analysis of the Privacy and Security Risks of Android VPN Permission-enabled Apps. In *Proceedings of the 2016 ACM on Internet Measurement Conference, IMC 2016, Santa Monica, CA, USA, November 14-16, 2016*. 349–364. http://dl.acm.org/citation.cfm?id=2987471

[29] VirusTotal Inc. 2019. VirusTotal Public API. https://www.virustotal.com/en/documentation/public-api/.

[30] InformAction. 2019. NoScript - JavaScript/Java/Flash blocker for a safer Firefox experience! - what is it? https://noscript.net. Accessed: 2019-08-09.

[31] Luca Invernizzi, Paolo Milani Comparetti, Stefano Benvenuti, Christopher Kruegel, Marco Cova, and Giovanni Vigna. 2012. Evilseed: A guided approach to finding malicious web pages. In *2012 IEEE symposium on Security and Privacy*. IEEE, 428–442.

[32] Sequa Jerome. 2019. Large Angler Malvertising Campaign Hits Top Publishers. https://blog.malwarebytes.com/threat-analysis/20/16/03/large-angler-malvertising-campaign-hits-top-publishers/. Accessed: 2019-01-18.

[33] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. 2015. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*. ACM, 45–56.

[34] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. 2015. Cutting the gordian knot: A look under the hood of ransomware attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[35] Deepak Kumar, Zane Ma, Ariana Mirian, Joshua Mason, J Alex Halderman, and Michael Bailey. 2017. Security Challenges in an Increasingly Tangled Web. In *Proceedings of the 2017 World Wide Web Conference on World Wide Web*.

[36] John Kurkowski. 2019. Accurately separate the TLD from the registered domain and subdomains of a URL, using the Public Suffix List. https://github.com/john-kurkowski/tldextract.

[37] Malwarebytes Labs. 2017. Malvertising on Equifax, TransUnion tied to third party script (updated). https://blog.malwarebytes.com/threat-analysis/2017/10/equifax-transunion-websites-push-fake-flash-player/.
[38] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *NDSS*.
[39] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. 2012. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 674–686.
[40] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. 2012. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 674–686.
[41] Rahat Masood, Dinusha Vatsalan, Muhammad Ikram, and Mohamed Ali Kaafar. 2018. Incognito: A Method for Obfuscating Web Data. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 267–276. https://doi.org/10.1145/3178876.3186093
[42] Mozilla. 2019. Cross-Origin Resource Sharing (CORS) - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.
[43] Arvind Narayanan and Vitaly Shmatikov. 2008. Robust De-anonymization of Large Sparse Datasets. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*. 111–125. https://doi.org/10.1109/SP.2008.33
[44] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are What You Include: Large-scale Evaluation of Remote JavaScript inclusions. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. 736–747. https://doi.org/10.1145/2382196.2382274
[45] Giancarlo Pellegrino, Christian Rossow, Fabrice J. Ryba, Thomas C. Schmidt, and Matthias Wählisch. 2015. Cashing Out the Great Cannon? On Browser-Based DDoS Attacks and Economics. In *9th USENIX Workshop on Offensive Technologies, WOOT '15, Washington, DC, USA, August 10-11, 2015*. https://www.usenix.org/conference/woot15/workshop-program/presentation/pellegrino
[46] Bogdan Popa. 2017. 85 Infected Android Apps Stealing Social Network Passwords Found on Play Store. https://news.softpedia.com/news/85-infected-android-apps-stealing-social-network-passwords-found-on-play- store-518984.shtml.
[47] Fabian Schneider, Sachin Agarwal, Tansu Alpcan, and Anja Feldmann. 2008. The New Web: Characterizing AJAX Traffic. In *Passive and Active Network Measurement, 9th International Conference, PAM 2008, Cleveland, OH, USA, April 29-30, 2008. Proceedings*. 31–40. https://doi.org/10.1007/978-3-540-79232-1_4
[48] SecurityWeek. 2017. Malicious Redirects on Equifax, TransUnion Sites Caused by Third-Party Scripts. https://www.securityweek.com/malicious-redirects-equifax-transunion-sites-caused-third-party-script.
[49] Jingxiu Su, Zhenyu Li, Stéphane Grumbach, Muhammad Ikram, Kavé Salamatian, and Gaogang Xie. 2018. Web Tracking Cartography with DNS Records. In *37th IEEE International Performance Computing and Communications Conference, IPCCC 2018, Orlando, FL, USA, November 17-19, 2018*. 1–8. https://doi.org/10.1109/PCCC.2018.8710841
[50] Jingxiu Su, Zhenyu Li, Stéphane Grumbach, Muhammad Ikram, Kavé Salamatian, and Gaogang Xie. 2019. A Cartography of Web Tracking using DNS Records. *Computer Communications* 134 (2019), 83–95. https://doi.org/10.1016/j.comcom.2018.11.008
[51] Mozilla Public Suffix. 2019. View the Public Suffix List. https://publicsuffix.org/list/.
[52] Latanya Sweeney. 1997. Weaving technology and policy together to maintain confidentiality. *The Journal of Law, Medicine & Ethics* 25, 2-3 (1997), 98–110.
[53] Ashlee Vance. 2009. Times web ads show security breach. https://www.nytimes.com/2009/09/15/technology/internet/15adco.html.
[54] Quick Remove Virus. 2017. How Do I Remove HWCDN.NET from My PC. https://quickremovevirus.com/how-do-i-remove-hwcdn-net-from-my-pc/.
[55] volatilityfoundation. 2019. volatilityfoundation/volatility: An advanced memory forensics framework. https://github.com/volatilityfoundation/volatility. Accessed: 2019-08-09.
[56] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*. 473–485. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/wang_xiao
[57] Zhaohua Wang, Zhenyu Li, Minhui Xue, and Gareth Tyson. 2020. Exploring the Eastern Frontier: A First Look at Mobile App Tracking in China. In *21st Passive and Active Measurement Conference (PAM)*.
[58] Websense. 2018. Real-time Threat Analysis with CSI: ACE Insight. https://csi.websense.com/.
[59] Benjamin Zi Hao Zhao, Muhammad Ikram, Hassan Jameel Asghar, Mohamed Ali Kaafar, Abdelberi Chaabane, and Kanchana Thilakarathna. 2019. A Decade of Mal-Activity Reporting: A Retrospective Analysis of Internet Malicious Activity Blacklists. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (Asia CCS '19)*. Association for Computing Machinery, New York, NY, USA, 193âĂŞ205. https://doi.org/10.1145/3321705.3329834

Received February 2019; revised August 2019; accepted January 2020