

# The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading\*

Muhammad Ikram  
muhammad.ikram@mq.edu.au  
Macquarie University  
University of Michigan

Rahat Masood  
rahat.masood@data61.csiro.au  
UNSW and Data61, CSIRO

Gareth Tyson  
g.tyson@qmul.ac.uk  
Queen Mary University of London

Mohamed Ali Kaafar  
dali.kaafar@mq.edu.au  
Macquarie University and Data61,  
CSIRO

Noha Loizon  
noha.loizon@data61.csiro.au  
Data61, CSIRO

Roya Ensafi  
ensafi@umich.edu  
University of Michigan

## ABSTRACT

The Web is a tangled mass of interconnected services, where websites import a range of external resources from various third-party domains. However, the latter can further load 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 of 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 did 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. By running sandboxed experiments, we observe a range of activities with the majority of suspicious JavaScript downloading malware; worryingly, we find this propensity is greater among implicitly trusted JavaScripts.

## ACM Reference Format:

Muhammad Ikram, Rahat Masood, Gareth Tyson, Mohamed Ali Kaafar, Noha Loizon, and Roya Ensafi. 2019. The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading. In *WWW'19: The Web Conference, May 13–15, 2019, San Francisco, USA*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

\*This is a preliminary (and extended) version of the paper with the same title which to appear in the proceedings of World Wide Web (WWW) 2019: The 2019 Web Conference, May 13-17, 2019.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

WWW'19, May 2019, San Francisco, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>

## 1 INTRODUCTION

In the modern web ecosystem, websites often load resources from a range of third-party domains such as ad providers, tracking services, content distribution networks (CDNs) 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, which loads JavaScript from the `widgets.com` domain, 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 (see Figure 1). Past work tends to ignore this, instead, collapsing these levels into a single set of third-parties [9, 29]. 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. The dynamic nature of the content being loaded and the wide adoption of in-path traffic alterations [6, 32] further complicates the issue. This potential threat should not be underestimated as errant active content (e.g., JavaScript) opens the way to a range of further exploits, e.g., Layer-7 DDoS attacks [30] or massive ransomware campaigns [20].

This paper studies dependency chains in the web ecosystem. Although there has been extensive work looking at the presence of third-parties in general [9, 27, 29], little work has focused on how content is indirectly loaded by first-party websites via dependency chains. We start by inspecting how extensive dependency chains are across the Alexa's top-200K (Section 2) [7]. We confirm their prominence, finding that around 50% of websites *do* allow third-parties to form dependency chains (i.e., they implicitly trust third-parties they do not directly load). The most commonly *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. Using the VirusTotal service [18] API, we classify third-party domains into innocuous vs. suspicious. When using a reasonable classification threshold, 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 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.

The prevalence of JavaScript resources being classified as suspicious leads us to further explore their activities. We therefore proceed to *sandbox* all suspicious JavaScript programs to monitor their activities (Section 5). We build a sandbox and perform tests executing suspicious JavaScript. We observe that the further down the dependency chain, the more active the suspicious JavaScript are with a high number of HTTP requests generated by suspicious JavaScript programs at level  $\geq 2$ . This is worrying as resources loaded further down the dependency chain are the most opaque to the website operator. We find evidence of involvement of first-party websites in malicious SEO (search) poisoning activities when (implicitly) loading some suspicious JavaScript content. Perhaps more importantly, we find that the most typical purpose of the suspicious JavaScript code is downloading dropfiles: again, the propensity to download files actually increases further along the chain with the most active JavaScript at level 4 downloading 129 files. We share our datasets, experimental testbed code and scripts used in this paper with the wider research community for further analysis of the consequences of the Web's implicit trust on <https://wot19submission.github.io>. We conclude the paper by summarising reality of a very fragile web ecosystem, revealing that suspicious parties within the dependency chains are relatively commonplace (§7).

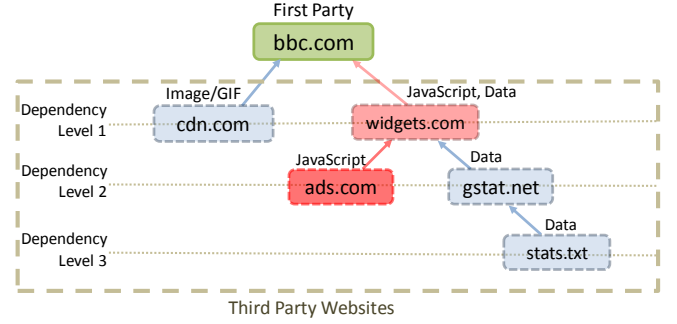
## 2 DATASET AND DATA ENRICHMENT

### 2.1 Alexa dependency dataset

We start by presenting our data collection methodology, and how we have validated its correctness.

**2.1.1 Data Collection.** We obtain the resource dependencies of the Alexa top-200K websites' main pages<sup>1</sup> using the method described in [24]. This Chromium-based Headless [12] 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

<sup>1</sup>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.



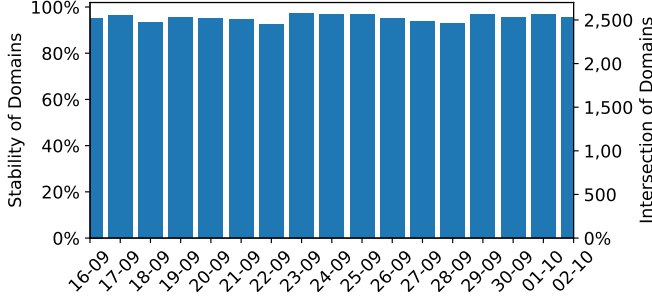
**Figure 1: Example dependency chain, including malicious third-party (in red).**

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 so that a request to a domain such as `player.bbc.com` is not considered as third-party. Due to the lack of purely automated mechanism to disambiguate between site-specific sub-domains (e.g., `player.bbc.com`) or country-specific sub-domains (e.g., `bbc.co.uk`), we leverage Mozilla Public Suffix list [35] and `tlldextract` [25] 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.

We acknowledge that the constructing the dependencies between objects in webpages is non-trivial task. In cases like where third-party JavaScript 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 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. Unfortunately, performing daily crawls for the Alexa top-200k websites was not possible. We therefore selected 1,500 domains as a seed for the crawler. This list consisted 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 daily a basis from September 15–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: Stability of day-by-day dependency trees analyzed per domain**

Figure 2 presents the day-to-day stability of the domains we see within each website.<sup>2</sup> We observe that 95.07% of second level domains remain consistent across consecutive days, and only an average 4.93% of domains were 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 strong indicator that utilizing a single snapshot is sufficient for gaining vantage into the use of third-parties. We leave inspection of temporal dynamics to our future work.

## 2.2 Data Enrichment

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. Therefore, it is imperative to rely upon multiple AV scanners and datasets to effectively classify domains as innocuous vs. suspicious. We leverage the capabilities offered by VirusTotal’s public API to automatize our classification process. VirusTotal is an online solution which aggregates the scanning capabilities provided by more than 68 AV tools, scanning engines and datasets. It has been commonly used in the academic literature to detect malicious apps, executables, software and domains [17, 21, 22].

We use the VirusTotal report APIs to obtain the VTscore for each third-party URL. Concretely, this score is 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 [40] category 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.

<sup>2</sup>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,  $\text{stability} = \frac{C_n \cap C_{n+1}}{C_n \cup C_{n+1}}$ .

## 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.

### 3.1 Do websites rely on implicit trust?

	Alexa Rank					
	1-200K	1-10K	190-200K	10-50K	50-100K	100-200K
<b>F.-Parties that trust:</b>						
<b>All Resources:</b>						
Explicit (Lvl. 1)	95%	95%	95%	94%	95%	95%
Implicit (Lvl. $\geq 2$ )	49.7%	55.1%	47.9%	51.8%	50.23%	48%
<b>JavaScript:</b>						
Explicit	91%	92%	91%	91%	91%	90%
Implicit	49.5%	55%	47.8%	51.69%	50%	47.8%

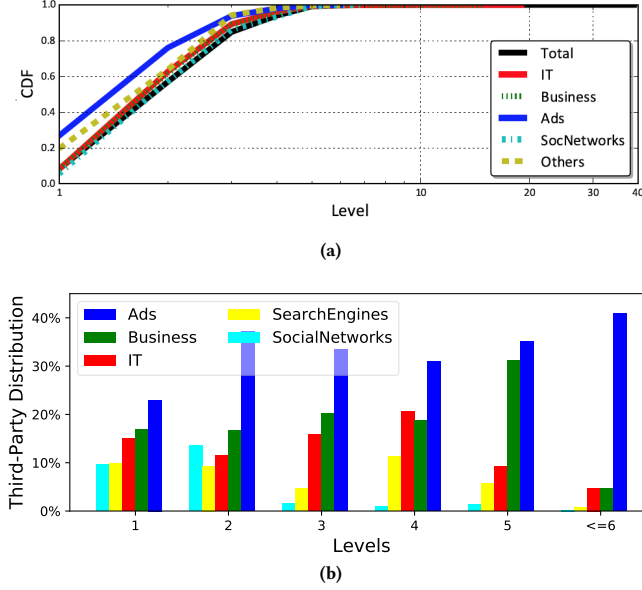
**Table 1: Overview of the Dataset for different ranges of Alexa’s ranking. The rows indicate the proportion of Alexa’s Top-X websites that explicitly and implicitly trust at least one third-party (i) resource (of any type); and (ii) JavaScript.**

Overall, the Top-200k dataset collectively 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 several prior studies [9, 27], it shows that 95% of websites import external resources, with 91% importing externally hosted JavaScript. This trend is already well known; more important is that observation that 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. 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 (*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 [39] and increase attacks surface. Figure 3a presents the CDF of chain length for all first-party websites. For context, websites are separated into their sub-categories.<sup>3</sup> 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  $\approx 2\%$  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.bg* (Harvard’s student newspaper), *argumenti.ru* (news), *mundomax.com* (IT news), *lifestyle.bg* (entertainment) 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.

Briefly, we also note that Figure 3a reveals subtle differences *between* different categories of third-party domains. For example,

<sup>3</sup>We only include the most popular categories.



**Figure 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.**

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 third-parties (e.g., Facebook plug-ins) are least likely to be loaded at level 1.

### 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, Data (consisting of HTML, JSON, XML, plain text files), and CSS/Fonts. 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 tend to increase — 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 compared to just 12.3% at level 3. This trend is caused by the fact that new levels are typically created by JavaScript execution (thus, by definition, the fraction of JavaScript must 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.

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 other highly demanded third-party categories such as search engines, Business and IT. These are led by well

Lev.	Total	Image	JS	Data	Font/CSS	Uncat.
1	9,212,245	34.4%	30.6%	16.0%	7.8%	11.3%
2	1,566,841	48.8%	16.7%	11.7%	3.3%	19.4%
3	405,390	45.0%	12.3%	11.1%	1.3%	30.2%
4	78,107	41.8%	18.4%	8.0%	8.1%	23.6%
5	14,413	40.6%	18.0%	12.8%	2.0%	26.4%
≥6	10,208	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.**

known providers, e.g., google-analytics.com (web-analytics<sup>4</sup>) 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, 46.2% at level 3, i.e., 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 resources. Naturally, these are led by major providers. 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.

## 4 FINDING SUSPICIOUS CHAINS

The previous section has shown that the creation of dependency chains is widespread, and there is therefore 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).

### 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  $\geq 1$ , as this indicates the remaining 67 virus checkers did not flag the domain<sup>5</sup>. Thus, we start by inspecting the presence of suspicious third-parties using a range of thresholds.

Table 3 shows the fraction of third-parties that are classified as suspicious using several VTscore thresholds. For context, we

<sup>4</sup>Grouped as in business category as per VirusTotal reports.

<sup>5</sup>Diversity is likely caused by the virus databases used by the different virus checkers [5]

Category	Third-Parties	Total Calls	Suspicious JS	VTscore $\geq 3$		VTscore $\geq 10$		VTscore $\geq 20$		VTscore $\geq 40$		VTscore $\geq 55$	
				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%	$\leq 0.1\%$	$\leq 0.1\%$
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%	$\leq 0.1\%$	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%	$\leq 0.1\%$	$\leq 0.1\%$
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%	$\leq 0.1\%$	$\leq 0.1\%$

**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. 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.,  $\geq 3$  and  $\geq 55$ ).**

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.

Additionally, we inspect first-party domains that inherit suspicious JavaScript resources from the explicit and various implicit levels. We focus on JavaScript as active web content that poses great threats with significant attack surfaces consisting of vulnerabilities related to client-side JavaScript, such as cross-site scripting (XSS) and advanced phishing [27]. Table 4 shows 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 no one category of domains that inherits suspicious JavaScript. However, we note that first party websites categorized as “Business” represent the majority of most exposed domains at Level  $\geq 2$ , with 16% of the total number of first-party domains implicitly trusting suspicious JavaScript belonging to the Business Category, with distant second being the “News & Media” Category and third the “Adult” category. The number of suspicious JavaScript codes loaded by these first-party domains ranges from 4 to 31 programs. We note the extreme case of *amateur-fc2.com* website *implicitly* importing 31 unique suspicious JavaScript programs from 4 unique suspicious domains. Moreover, we observe at most 7 unique third-parties (combining both explicit and implicit level) that is a cause of suspicious JavaScripts in first-parties. This happens for *privet-rostov.ru* domain, having third-party domains such as *charter.com*, *vk.com*, *rambler.ru*, *doubleclick.net*, *dx.com*, *cdn.adlegend.com*, *syncsw.pool.datamind.ru*.

## 4.2 How widespread are suspicious parties?

We next inspect how “popular” 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

Unique Suspicious Domains at Level = 1						
#	First-party Domain	Alexa Rank	# Mal. JSes	Unique Susp. Doms.	Category	Chain Len.
1	theinscribermag.com	46,242	6	5	Blogs	5
2	skynet-system.com.ua	192,549	6	5	Busin.	4
3	nodwick.com	194,823	13	4	Enter.	4
4	iphones.ru	12,045	4	4	IT	4
5	privet-rostov.ru	193,024	6	4	LifeStyle	4

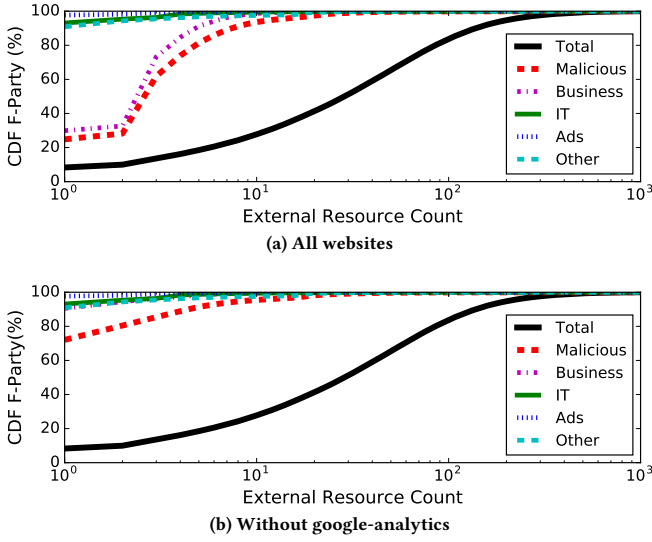
Unique Suspicious Domains at Level $\geq 2$						
1	traffic2bitcoin.com	33,513	6	5	Games	7
2	radionetplus.ru	166,003	8	4	SW Download	6
3	studiofow.tumblr.com	85,483	11	4	Adult	4
4	amateur-fc2.com	52,556	31	4	Adult	5
5	fasttorrent.ru	24,250	9	4	File Sharing	7

**Table 4: Top 5 most exposed first-party domains (with VTscore  $\geq 10$ ) ranked by the number of unique suspicious domains.**

(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  $\geq 10$ . The figure 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 third-party domains are classified as suspicious, their reach covers nearly three quarters of websites (indirectly via implicit trust).

This is a product of the power-law distribution of third-party “popularity” across websites: 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. Hence to understand the impact its new de-classification has, Figure 4b shows the distribution of resource calls to third-party categories





**Figure 4: CDF of resources loaded per-website from various categories of third-parties.**

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.

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

**Table 5: Top 5 most prevalent suspicious third-party domains (with VTscore  $\geq 10$ ) on level 1 (explicit trust) and beyond (implicit trust) providing resources to first-parties. #FP refers to the number of First-party domains having the corresponding suspicious third-party domain in their chain of dependency.**

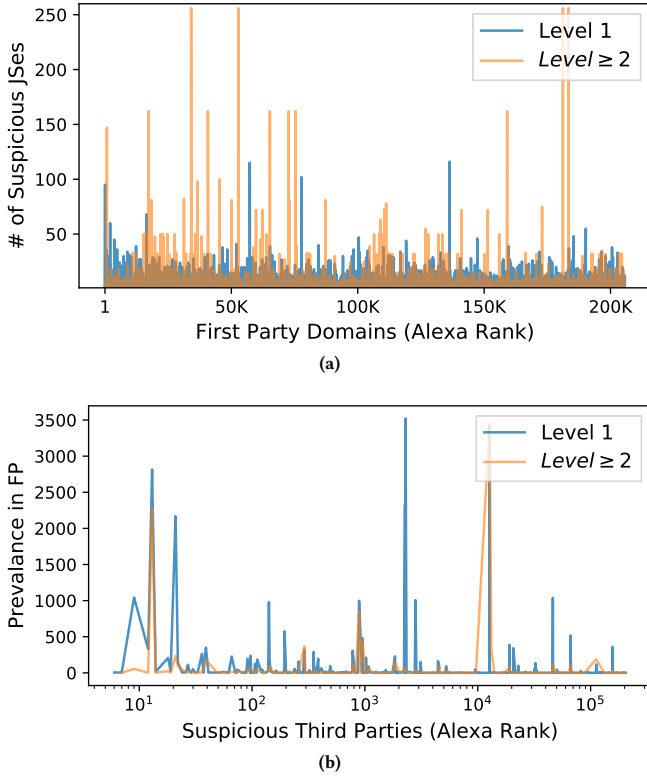
Next we inspect in, Table 5, the top 10 most frequently encountered suspicious third-party domains that are providing suspicious JavaScript resources to first-parties (as opposed to the most exposed first-party domains shown earlier in Table 4). We rank these suspicious third-party domains according to their prevalence in the Web ecosystem and further decompose our analysis at explicit and implicit levels in the table.

As already discussed, we found `google-analytics.com` among the most called domains. Interestingly, we find several 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  $\geq 2$ . An obvious reason for this domain’s presence is because of other infected (malware-based) apps that try to authenticate users from such domains [31]. 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 [8]. For instance, users in `statcounter` forums reported it as malicious because a Javascript running its website redirects users to a malware website `gocloudly.com`, and forces users to click the button [10].

While it is not shown in the tables, we also note the presence of `qq.com`, a Chinese Search Engine ranked high by Alexa. This is among the top 10 most encountered suspicious third-party domains, as defined by several AntiVirus tools within VirusTotal. Closer inspection reveals this is likely due to repeated instances of insecure data transmission, use of qq fake accounts for malware manifestation and for data encryption Trojans [13, 23, 38].

More generally, we observe the presence of a wide range of Alexa ranks in the list of most prevalent domains at levels  $\geq 2$ . In Figure 5a, we show the number of suspicious JavaScripts 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  $\geq 2$ , however, the first-party domains seem to be equally vulnerable to the implicit import of suspicious content regardless of their rank. There are exceptions though, signified by the peaks in the number of suspicious JavaScripts — these are near exclusively driven by a large number of  $\geq$ 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 [37]. 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. We observe that some very prevalent third-parties have a high Alexa ranking (even excluding `google-analytics.com` which as per our previous observation has the highest prevalence of 43,156 impacted first-party domains and hence excluded from Figure 5b for readability purposes). Note a spike around 2000 rank, which reaches a prevalence of 3500 first-party domains at level 1. This spike is caused by `gravater.com`, propagating suspicious Javascripts. This supports our statements earlier (from Table 5) where `gravater.com` is ranked 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



**Figure 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.

variety of third-party suspicious JavaScript content loaded from various, not necessarily “obscure”, third-party domains.

### 4.3 Which websites are impacted?

Next, we inspect the location(s) in the dependency chain where these suspicious third-parties are situated, as well as the *types* of websites that load them. This is vital, as implicitly trusted ( $\geq$  level 2) resources are far more difficult for a first-party administrator to remove — they could, of course, remove the intermediate level 1 resource, but this may disrupt their own business activities. Table 6 presents the proportion of websites that import at least one resource with a VTscore  $\geq 10$ . We separate resources into their level in the dependency chain. Interestingly, the majority of resources classified as suspicious are located 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 suggests that these website operators are not entirely diligent in monitoring their third-party resources. Perhaps more important, the above leaves a significant minority of suspicious resources imported via *implicit* trust (*i.e.*, level  $\geq 2$ ). 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  $\geq 10$ . 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 [28, 36].

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%
$\geq 5$	$\leq 0.1\%$	0	0.002	$\leq 0.1\%$	0.001%	$\leq 0.1\%$	0.002%	$\leq 0.1\%$	$\leq 0.001\%$	0.00%

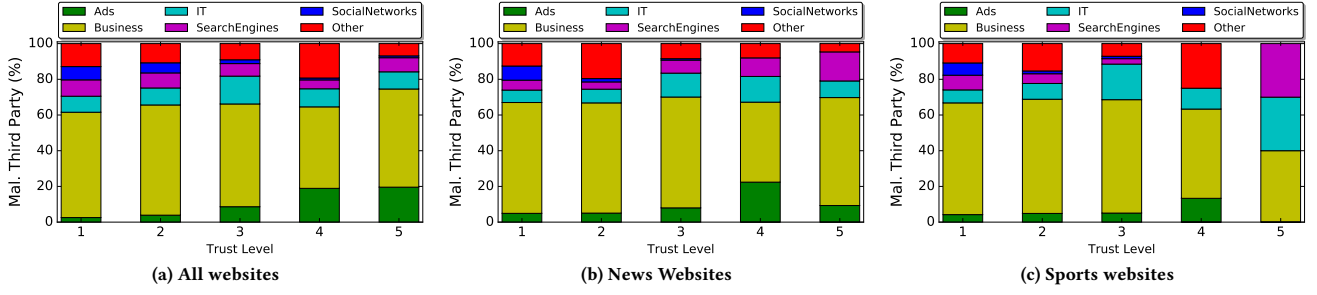
**Table 6:** Proportion of top-200K websites importing resources classified as suspicious (with VTscore  $\geq 10$ ) at each level.

Similar, albeit less extreme, observations can be made across Sports, Entertainment, and Forum websites. Briefly, Figure 6 also 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 `convexty.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).

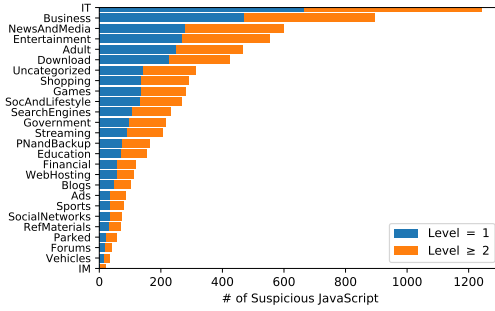
Next, we again focus on JavaScript content as, when loaded, it can represent significant security risks: Our analysis is motivated by well known attack vectors underpinned by JavaScript, *e.g.*, malvertising [28], malware injection and exploit kits redirection. These are exemplified by the recent reporting that Equifax and TransUnion were hit by a third-party web analytics script [26, 34]. Figure 7 presents the breakdown of the domain categories hosting the JavaScript resources we observe. Clear trends can be seen, with IT (*e.g.*, `dynaquestpc.com`), Business (*e.g.*, `vindale.com`), News and Media (*e.g.*, `therealnews.com`), and Entertainment (*e.g.*, `youwatchfilm.net`) dominating.

Clearly, suspicious JavaScript resources cover a broad spectrum of activities. Interestingly, we observed that 70% and 67%, respectively, of Business (Web analytics) and Ads JavaScripts are loaded from level  $\geq 2$  in contrast to 17% and 31% of JavaScripts of Government and Shopping loaded at level 1. We next strive to quantify the level of suspicion raised by each of these JavaScripts. 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 JavaScripts into their location in the dependency chain. Clear difference can be observed, with level 2 obtaining the highest VTscore (median 28). In fact, 78% of the suspicious JavaScript resources loaded on trust level 2 have a VTscore  $> 52$  (indicating very high confidence).

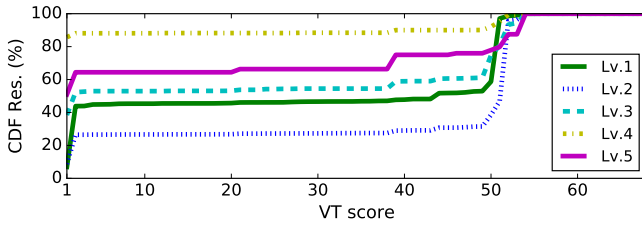
This is a critical observation since as mentioned earlier, while suspicious third-parties at level 1 can be ultimately removed by first-party website operators if flagged as suspicious, this is much more difficult for implicitly trusted resources further along the dependency chain. If the intermediate (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



**Figure 6: Distribution of calls to 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).**



**Figure 7: Breakdown of JavaScript resources based on category of domain. Uncategorized category includes domain such as newmyvideolink.xyz and cooster.ru**



**Figure 8: CDF of suspicious JavaScripts (VTscores ≥ 10) at different levels in the chain.**

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

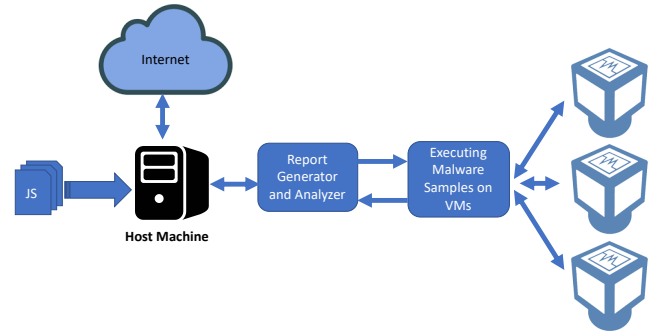
## 5 ANALYSIS OF SUSPICIOUS JAVASCRIPT RESOURCES

JavaScript is arguably the riskiest resource to import as this has the potential to execute diverse functions (including the downloading

of further resources). Thus, we next inspect the activities of the 7,166 JavaScript scripts that were classified as suspicious.

### 5.1 Methodology

We use a dedicated testbed, depicted in Figure 9, composed of three Virtual machines (VMs) that connect to the Internet via a computer that runs Cuckoo sandbox and tcpdump, respectively, to log all system-level events and to intercept all the traffic being transmitted between the virtual machines and the Internet. These VMs monitor already detected publishers using different browser user-agent configurations (IE 11 and Firefox 63.0.1) and cookie clearing strategies (“always clear cookies”). This allows us to observe the traffic generated by each JavaScript code when it is rendered by the VMs’ browsers. For instance, Cuckoo sandbox keeps track of network traffic generated; all types of file operations, memory changes, registry changes when the JavaScript code was loaded.



**Figure 9: Overview of our testbed with a host machine running Cuckoo sandbox (v2.0.6).**

For each test, first we create an HTML code and inject suspicious JavaScript code in `<script>` tag and then load the HTML code in the browser of one of our VMs. Each test lasts  $123.01 \pm 2.53$  seconds. Prior to each test, we also ensure that the previous JavaScript code we experimented with is removed from the browser history as well as browser and VM’s cache (e.g., DNS cache) and we reboot the device to enforce the complete renewal of the VM. This analysis draws



boundary for the suspicious behavior of JavaScript code during suspicious redirects and local file changes. We share the code and data for the testbed is at <https://wot19submission.github.io>.

## 5.2 Results

Our sandbox provides vantage onto the network activity generated by JavaScript resources, alongside any dropfile they generate. We next analyze the JavaScript along these two axes.

**5.2.1 Summary of Suspicious JavaScripts.** We begin by providing a brief overview of the most prominent JavaScript resources observed in our dataset. 44.7% exist at level 1 (explicit trust) and 55.3% at level  $\geq 2$  (implicit trust). Table 7 provides a list of the JavaScript resources that generate the most HTTP requests. We separate them into implicit and explicitly trusted resources. The most typical purpose of these JavaScripts are downloading dropper files. Whereas those at level 1 tend to have lower VTscores, whereas VT classifies those at level  $\geq 2$  with a higher confidence. The most regularly observed JavaScript at level 1 is `new-play-1.js`, a relatively highly ranked (22,574 Alexa) script which downloads dropfiles—executables such as Exploitkits, Trojans, Fake AV or mediaplayers performing suspicious activities. In contrast, at level  $\geq 2$ , it is `blog.js`, which show annoying ads and involve in click fraud. Overall, we see that 99.5% of JavaScript resources download dropfiles with 98.62% of it involved in malvertising and click frauds.

**5.2.2 Analyzing Network Activity.** We observe significant network activity generated by the suspicious JavaScript resources within our testbed. Figure 10 presents the CDFs of the number of HTTP requests generated per suspicious JavaScript code. The figure splits the JavaScript into their respective positions in the dependency chains, and their categories (categories of third-party domains in which they originate from). 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  $\geq 4$  are extremely active (median 30). 36% of the JavaScripts imported from level 5 generate at least 30 HTTP requests in contrast to 15% of the JavaScripts sourced from level 2. This is in contrast to a typical behavior of legitimate JavaScripts that have been previously measured to generate on average 4 HTTP requests[33]. In essence, we observed that the further down the dependency chain, the more active the suspicious JavaScript. This is worrying as resources loaded further down the dependency chain are the most opaque to the website operator.

Figure 10b and 10c also separate the JavaScript resources into their respective categories. Whereas those at level 1 (explicit trust) exhibit relatively similar traits across all categories, we find that those at level  $\geq 2$  (implicit trust) have far more divergence. Those classified as Business, IT or Adult are the most active, whereas News, Ads and Download generate the fewest. This is driven by the fact that most Business (*i.e.*, subcategory web-analytics) domains are continuously tracking dynamic content for a number of domains rather than, for example, JavaScripts codes from IT (*i.e.*, subcategory hosting providers), Adult, and Blogs domains, which source static resources for various domains [16]. For instance, JavaScripts from Business category at level 1 generate on average 3 HTTP requests vs. 19 for those at level  $\geq 2$ .

A very important consideration here is to understand what are the domains that are targeted by these HTTP requests. 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 11 presents a heatmap illustrating the number of requests to the top 25 domains by the suspicious JavaScript codes, with the X-axis showing the suspicious JavaScript, the Y-axis listing the targeted domains and the heat defined by the fraction of requests that each JavaScript issued to each domain. We immediately see that most JavaScripts have a distinct preference towards a small set of domains. 18% of JavaScripts submit over 50% of their HTTP requests to a single domain. One particularly popular domain is `bing.com`; 65% of all JavaScripts access this domain at least once. A closer inspection of these JavaScript suggests that most of the ones targeting `bing.com` undertake some form of SEO poisoning (search poisoning) activities launching exploits to trick and elevate the ranking of some particular URLs in the results listings of the search engine [15, 19].

Figure 12 presents the count of HTTP requests across the top 25 targeted domains. We separate into JavaScripts at level 1 vs. level  $\geq 2$ . Again, the figure shows that most commonly occurring HTTP fetch is to `bing.com`; In general, we observed that search engines are heavily targeted by scripting codes (to elevate certain URLs and websites as observed in previous research [19]). Similar organizations are seen within this top ranking, *e.g.*, `microsoft.com`, `google.com`. We stress the importance of this observation as we recall that the JavaScript code is loaded on the first-party domain (either explicitly or implicitly). Hence the first-party domains might be involved in stealthy suspicious activities triggered by third-party domains the former are not even aware of. This observation is even more enlightened by our earlier observation that suspicious JavaScript are more active when at level  $\geq 2$  which is confirmed in Figure 12. We note the existence of various certification authorities and observed various JavaScript codes requesting encryption (SSL) methods and HTTPS requests that we did not investigate further. We leave this as future work.

**5.2.3 Analyzing Dropfiles.** The above has revealed that a large number of suspicious JavaScript resources download files. The use of “droppers” 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.

We find that 99.5% of the analyzed JavaScript resources generate dropfiles, indicating that this is one of the most common used behavior of suspicious JavaScripts. Figure 13 depicts the distribution of the number of files dropped by the JavaScript content. We observe a significant number of active memory operations as depicted by the number of dropped files: remarkably, 22% of JavaScript generate at least 8 files from memory. Table 8 presents the top-10 most active, ranked according to the number of dropfiles, suspicious JavaScript. Some of these JavaScript resources are extremely active. For example, `xbigg.com/adv.js`, which is loaded at level 1, downloads 16 files. More worryingly, the propensity to download drop files actually increases further along the chain: the most active JavaScript, `http://yourjavascript.com/3439241227/blog.js` at level

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

Table 7: Top 5 suspicious JSes, with at least 41 HTTP requests, found in the dependency chain (level 1 and levels  $\geq 2$ ) of 200K domains.

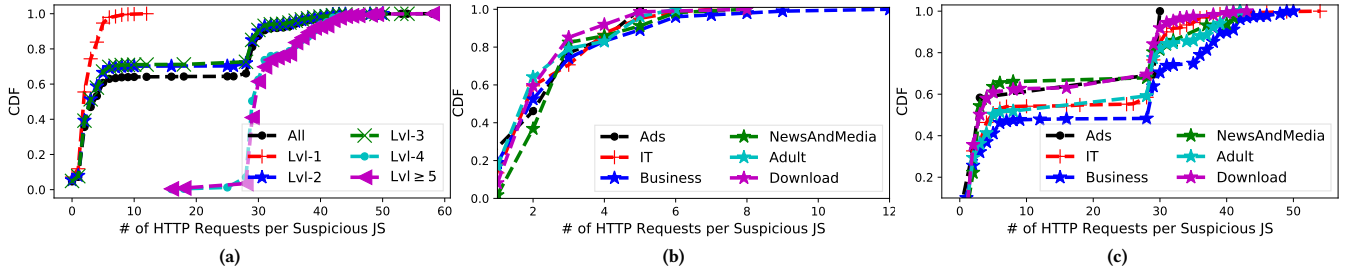


Figure 10: CDFs of number of HTTP requests generated per suspicious JS viewed across categories of domains and dependency levels.

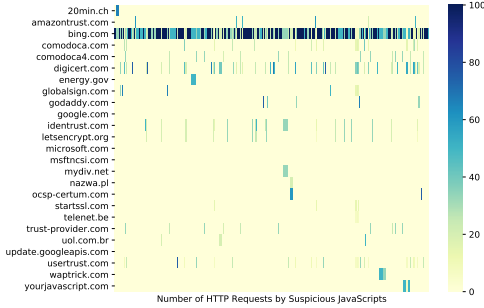


Figure 11: Heatmap of number of requests to domains by suspicious JavaScript codes and histogram of top 25 contacted domains by suspicious JavaScripts.

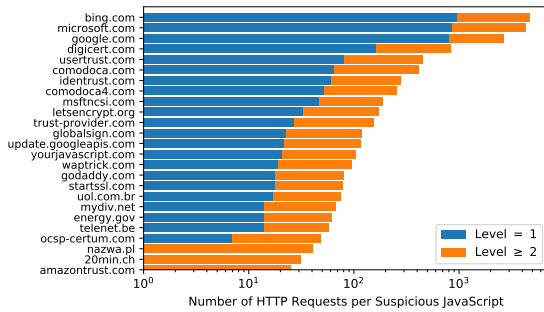


Figure 12: Number of HTTP requests launched to each domain.

4 downloads 129 files. It is also interesting to observe that the resources at level  $\geq 2$  also 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. 8% are encrypted, and therefore cannot be examined. The remainder are Potentially Unwanted Programs (0.52%), Exploitkits (0.36%), Adware and Click Bots (98.62%), and Trojan (0.50%) according to VirusTotal reports. For instance, we observe that videowood.tv/assets/js/poph.js uses eval() –JavaScript’s dynamic loading method– to download and executes via 1832-fc204a9bcefeab3d.exe (with VTscore=5)– to take over web browser for displaying a wide range of annoying ads and to garner fraudulent clicks.

## 6 RELATED WORK

There has been a wealth of research into the utilisation and exploitation of third-parties. Falahrestegar *et al.* [9] inspected the use of third-parties across top Alexa websites, exploring how third-party operators differ based on region. Nikiforakis *et al.* [29] demonstrated in 2012 that large proportions of websites rely on JavaScript libraries hosted on ill-maintained external web servers, making JavaScript exploits trivial. Lauinger *et al.* [27] led a further study, classifying sensitive libraries and the vulnerabilities caused by them. Gomer *et al.* [11] 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.* [14] 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

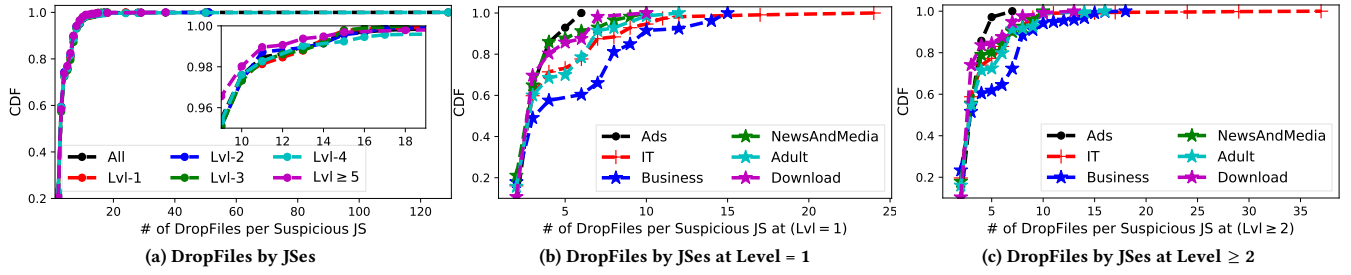


Figure 13: CDF of dropfiles downloaded and operated (i.e., read/write operation) by suspicious JavaScript codes.

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

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.* [24], 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.

## 7 CONCLUDING REMARKS

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. Of course, the most commonly *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 surfaces. 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 level2), where 78% of JavaScripts have a VTscore > 52. 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 sandboxes experiments on the suspicious JavaScript to understand their actions. We witnessed extensive download activities, much of which consisted of dropper files and malware, which was being installed on the machine. 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, SEO poisoning, malvertising and exploit kits 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. Without this, we are reticent to draw long-term conclusions. Another line of work is understanding how level ≥ 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. We intend to perform graph analysis to understand how removing these hypergiants may impact the presence of interconnected suspicious third-parties. Finally, by opening our collected datasets and experiments code and scripts to the wider research community, we hope this paper has shed some light on an important security consideration of today's Web and we call for further collaboration and research to help address this.

## REFERENCES

- [1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.
- [2] M. A. Bashir, S. Arshad, W. Robertson, and C. Wilson. Tracing information flows between ad exchanges using retargeted ads. In *USENIX Security Symposium*, 2016.
- [3] B. Bencs  th, G. P  k, L. Butty  n, and M. F  legyh  zi. Duqu: Analysis, detection, and lessons learned. In *ACM European Workshop on System Security (EuroSec)*, volume 2012, 2012.
- [4] A. Bertolino, G. Canfora, and S. G. Elbaum, editors. *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. IEEE Computer Society, 2015.
- [5] J. Canto, M. Dacier, E. Kirda, and C. Leita. Large scale malware collection: lessons learned. In *IEEE SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems*. Citeseer, 2008.
- [6] J. Chen, X. Zheng, H.-X. Duan, J. Liang, J. Jiang, K. Li, T. Wan, and V. Paxson. Forwarding-loop attacks in content delivery networks. In *NDSS*, 2016.
- [7] A. company. The top sites on the web. <http://www.alexa.com/topsites>, 2017.
- [8] I. X. Exchange. Statcounter session hijack. <https://exchange.xforce.ibmcloud.com/vulnerabilities/20506>, 2005.
- [9] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier. Anatomy of the third-party web tracking ecosystem. *Traffic Measurements Analysis Workshop (TMA)*, 2014.
- [10] S. C. Forum. <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/>, 2016.
- [11] R. Gomer, E. M. Rodrigues, N. Milic-Fraying, and M. Schrafel. Network analysis of third party tracking: User exposure to tracking cookies through search. In *WI-IAT*, 2013.
- [12] Google. Headless chromium. <https://chromium.googlesource.com/chromium/src/+lkgr/headless/README.md>, 2018.
- [13] A. GreenBerg. Hack brief: malware hits 225,000 (jailbroken, mostly chinese) iphones. <https://www.wired.com/2015/08/hack-brief-malware-hits-225000-jailbroken-mostly-chinese-iphones/>, 2015.
- [14] P. Holzinger, S. Triller, A. Bartel, and E. Bodden. An in-depth study of more than ten years of java exploitation. In *CCS*, 2016.
- [15] F. Howard and O. Komili. Poisoned search results: How hackers have automated search engine poisoning attacks to distribute malware. *Sophos Technical Papers*, pages 1–15, 2010.
- [16] M. Ikram, H. J. Asghar, M. A. Kaafar, A. Mahanti, and B. Krishnamurthy. Towards seamless tracking-free web: Improved detection of trackers via one-class learning. *Proceedings on Privacy Enhancing Technologies*, 2017(1):79–99, 2017.
- [17] M. Ikram, N. Vallina-Rodriguez, S. Seneviratne, M. A. Kaafar, and V. Paxson. An analysis of the privacy and security risks of android vpn permission-enabled apps. In *IMC*, 2016.
- [18] V. Inc. Virustotal public api. <https://www.virustotal.com/en/documentation/public-api/>, 2017.
- [19] L. Invernizzi, P. M. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and G. Vigna. Evilseed: A guided approach to finding malicious web pages. In *2012 IEEE Symposium on Security and Privacy*, pages 428–442. IEEE, 2012.
- [20] S. Jerome. Large angler malvertising campaign hits top publishers. <https://blog.malwarebytes.com/threat-analysis/2016/03/large-angler-malvertising-campaign-hits-top-publishers/>. Accessed: 2018-09-18.
- [21] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 45–56. ACM, 2015.
- [22] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer, 2015.
- [23] J. Knockel, A. Senft, and R. Deibert. Wup! there it is privacy and security issues in qq browser. <https://citizenlab.ca/2016/03/privacy-security-issues-qq-browser/>, 2016.
- [24] D. Kumar, Z. Ma, A. Mirian, J. Mason, J. A. Halderman, and M. Bailey. Security Challenges in an Increasingly Tangled Web. In *Proceedings of the 2017 World Wide Web Conference on World Wide Web*, 2017.
- [25] J. Kurkowski. Accurately separate the TLD from the registered domain and subdomains of a url, using the public suffix list. <https://github.com/john-kurkowski/tldextract>, 2018.
- [26] M. Labs. 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/>, 2017.
- [27] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. In *NDSS*, 2017.
- [28] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang. Knowing your enemy: understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 674–686. ACM, 2012.
- [29] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: Large-scale evaluation of remote javascript inclusions. In *CCS*, 2012.
- [30] G. Pellegrino, C. Rossow, F. J. Ryba, T. C. Schmidt, and M. W  dhlich. Cashing out the great cannon? on browser-based ddos attacks and economics. In *USENIX*, 2015.
- [31] B. Popa. 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>, 2017.
- [32] C. Reis, S. D. Gribble, T. Kohno, and N. C. Weaver. Detecting in-flight page changes with web tripwires. In *NSDI*, 2008.
- [33] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann. The new web: characterizing ajax traffic. In *International Conference on Passive and Active Network Measurement*, pages 31–40. Springer, 2008.
- [34] SecurityWeek. Malicious redirects on equifax, transunion sites caused by third-party scripts. <https://www.securityweek.com/malicious-redirects-equifax-transunion-sites-caused-third-party-script>, 2017.
- [35] M. P. Suffix. View the public suffix list. <https://publicsuffix.org/list/>, 2018.
- [36] A. VANCE. Times web ads show security breach. <https://www.nytimes.com/2009/09/15/technology/internet/15adco.html>, 2009.
- [37] Q. R. Virus. How do i remove hwcdn.net from my pc. <https://quickremovevirus.com/how-do-i-remove-hwcdn-net-from-my-pc/>, 2017.
- [38] Q. R. Virus. How to remove nintendonx@qq.com virus completely. <https://quickremovevirus.com/how-to-remove-nintendonxqq-com-virus-completely7>, 2017.
- [39] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystify page load performance with wprof. In *Proc. of the USENIX conference on Networked Systems Design and Implementation (NSDI)*, 2013.
- [40] Websense. Real-time threat analysis with csi: Ace insight. <https://csi.websense.com/>, 2018.