2015 IEEE Symposium on Security and Privacy

Ad Injection at Scale: Assessing Deceptive Advertisement Modifications

Kurt Thomas[◇], Elie Bursztein[◇], Chris Grier[□], Grant Ho[†], Nav Jagpal[◇], Alexandros Kapravelos[▽], Damon McCoy[‡†*], Antonio Nappa[§○], Vern Paxson[†*], Paul Pearce[†], Niels Provos[◇], Moheeb Abu Rajab[◇]

{*kurtthomas, elieb, nav, niels, moheeb*}*@google.com* {*grantho, vern, pearce*}*@cs.berkeley.edu*
antonio.nappa@imdea.org *chris@databricks.com* *damon@cs.gmu.edu* *kapravel@cs.ucsb.edu*

[◇]Google [†]University of California, Berkeley [*]International Computer Science Institute
[‡]George Mason University [□]Databricks [○]IMDEA Software Institute
[§]Universidad Politécnica de Madrid [▽]University of California, Santa Barbara

***Abstract*—Today, *web injection* manifests in many forms, but fundamentally occurs when malicious and unwanted actors tamper directly with browser sessions for their own profit. In this work we illuminate the scope and negative impact of one of these forms, *ad injection*, in which users have ads imposed on them in addition to, or different from, those that websites originally sent them. We develop a multi-staged pipeline that identifies ad injection in the wild and captures its distribution and revenue chains. We find that ad injection has entrenched itself as a cross-browser monetization platform impacting more than 5% of unique daily IP addresses accessing Google—tens of millions of users around the globe. Injected ads arrive on a client's machine through multiple vectors: our measurements identify 50,870 Chrome extensions and 34,407 Windows binaries, 38% and 17% of which are explicitly malicious. A small number of software developers support the vast majority of these injectors who in turn syndicate from the larger ad ecosystem. We have contacted the Chrome Web Store and the advertisers targeted by ad injectors to alert each of the deceptive practices involved.**

I. Introduction

With the advent of the cloud, web browsers now arbitrate access to a vast breadth of information, social interactions, and sensitive personal data stored remotely on the Internet. This evolution—where browsers are now analogous in function to operating systems—has lead to an entire new class of security threats facing users. Malicious and unwanted actors tamper directly with browser sessions to profit from: redirecting search traffic; inserting rogue tracking pixels; hijacking session cookies to spam email contacts and online social networks; and stealing personal and banking data. We refer to this broad category of threats as *web injection*.

Within this ecosystem, *ad injection* reigns as one of the most lucrative strategies for monetizing browser traffic. Popular examples include public WiFi portals that tamper with in-transit HTTP content to inject ads [7], [22]; and the Yontoo browser plugin which modified 4.5 million users' private Facebook sessions to include ads that earned Yontoo $8 million [21], [30]. These scenarios highlight that ad injectors skirt the line demarcating legitimately acquired traffic versus synthetic traffic generated via automated click fraud [8], click hijacking [1], [26], and impression fraud [32]. This distinction is critical—most ad injectors are *potentially unwanted programs*, not malware.

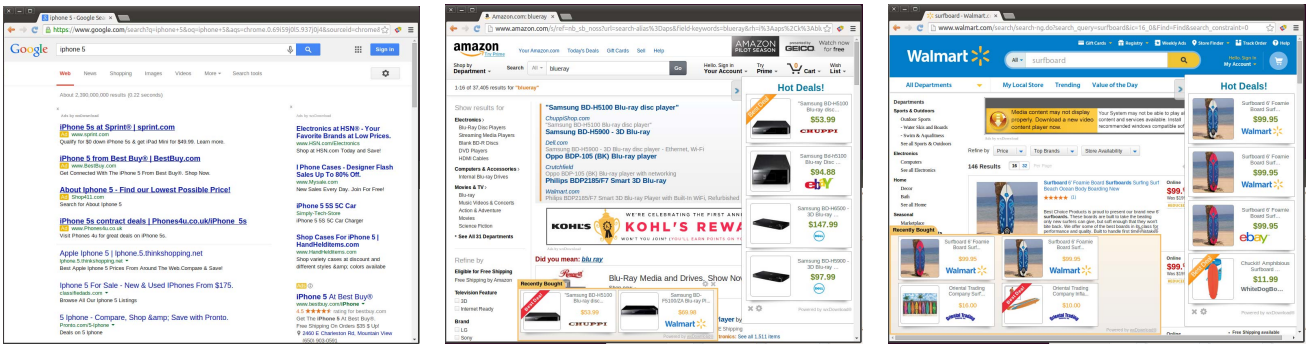
In this work we illuminate the negative impact of ad injection on users and expose the structure of the ad injection ecosystem. Of over 100,000 triaged Chrome user complaints in July, 2014, nearly 20% were related to ad injection—the single largest source of frustration. Our contributions consist of (1) measuring the volume of browser clients impacted by ad injection; (2) evaluating the relationship between ad injection and malicious or unwanted software; and (3) identifying the intermediaries and advertisers that support ad injection.

To conduct our study, we develop a multi-staged pipeline that captures ad injection's distribution and revenue chain. The pipeline starts with a client-side DOM scanner that identifies and reports the presence of rogue ad elements. We deploy this scanner on a subset of Google websites, collecting over 100 million client-side reports from a period spanning June 1–September 30, 2014. Next, we dynamically execute 25 million binaries and 1 million extensions in search of the same ad injectors we observe impacting clients. In the process, we analyze the techniques each ad injector uses to manipulate DOM content and identify the scope of properties that injectors affect. Finally, we dynamically execute a sample of injectors, automatically triggering the injector's logic to fetch ads and discover the intermediaries involved.

Our results reveal that ad injection has entrenched itself as a cross-browser monetization platform that impacts tens of millions of users around the globe. Our client-side telemetry finds that 5.5% of unique daily IP addresses visiting Google properties have at least one ad injector installed. The most popular, *superfish.com*, injects ads into more than 16,000 websites and grossed over $35 million in 2013 according to financial reports [16]. We find that all of the top ad injectors are organized as affiliate programs that decouple advertisement selection from third parties responsible for taking hold of a client's browser. We enumerate the top affiliates for each program and determine most are popular browser plugins such as ShopperPro, Yontoo, and PlusHD. Consequently, we find ad injection affects all prominent operating systems; we observe injections in 3.4% and 5.1% of pages served to Mac and Windows users, respectively.

Injected ads arrive on a client's machine through multiple unwanted and potentially malicious vectors. Our analysis pipeline flagged 50,870 Chrome extensions as unwanted ad

DOI 10.1109/SP.2015.17



(a) Google (b) Amazon (c) Walmart

Figure 1: Sample of ad injection on different search and shopping properties. None of the ads displayed are organic to the page.

injectors, 38% of which were outright malware. Extensions aggressively pursue injection profit vectors: 24% also spam Facebook and 11% hijack search queries. While Google previously disabled most of these deceptive extensions, we identified 192 with over 14 million users that were still active. We reported these to the Chrome Web Store, who confirmed they violated the Web Store's policies around deceptive ad injection[1] and subsequently disabled the extensions. A similar picture emerges for 34,407 Windows binaries we flagged for ad injection, 17% of which are malware. Many of these act as staged installers that in turn "side-load" extensions, while others install proxies that tamper with in-transit requests.

The ad injection ecosystem profits from over 3,000 advertisers including Sears, Walmart, Ebay, and Target, who unwittingly pay for traffic from injectors. These advertisers rarely have insights into the provenance of traffic because their perspective is limited to only the last hop in a convoluted web of intermediaries, which makes it difficult for brands to protect themselves from traffic sourced from ad injectors. Alternatively, we find that traffic enters the ad ecosystem through a small bottleneck of intermediaries run by ShopZilla, DealTime, and PriceGrabber. We are currently reaching out and alerting the advertisers and intermediaries impacted by traffic from ad injectors.

In summary, we frame our key contributions as follows:

- We develop a client-side technique that detects tens of millions of instances of ad injection impacting Google's users. Any website can re-use this technique.
- We conduct a detailed investigation of ad injection binary and extension delivery mechanisms.
- We detect and report 192 deceptive Chrome extensions impacting 14 million users; all have since been disabled. Our techniques for catching these extensions are now used by Google to scan new and updated extensions.
- We identify the bottlenecks in ad injector revenue chains and are reaching out to the advertisers and intermediaries impacted by the deceptive practices involved.

[1]We note that ad injection, when properly disclosed to users, is not explicitly prohibited by the Chrome Web Store.

II. Background

We begin by defining ad injection as used throughout this paper. We provide a number of real examples that capture the browsing experience of users impacted by ad injection before discussing the revenue model involved.

A. Characterizing Ad Injectors

We broadly refer to *ad injectors* as any binary, extension, or network ISP that modifies a page's content to insert or replace advertisements, irrespective of user consent. This definition notably excludes programs that remove advertisements (e.g., ad block software). Ad injectors can negatively impact a user's browsing experience, security, and privacy. We show that ad injectors frequently monitor *all* of a user's browser activities—including page interactions and search queries—and report these behaviors to third parties for tracking and advertisement selection. This process increases page load latency while injectors fetch numerous third-party scripts and generate XHR requests for ad content. Similarly, injectors degrade page quality by including spurious "search results," keyword highlights, or fly-in banners, many of which are irrelevant to the page and overwhelm the original content. Finally, as the provenance of ads is no longer controlled by a page's owner, injectors can expose users to irreputable intermediaries serving spam and malware. This tampering is invisible to the user making it appear as though the webpage was culpable rather than the ad injector, potentially degrading brand reputation.

B. Examples of Ad Injection User Experience

Once an ad injector takes hold of a client's browser session, a vastly different web experience emerges, a sample of which we show in Figure 1. None of the ads displayed in the figures—even those mimicking the page style—are organic; all originate from a single extension installed on the client's browser. The examples shown are particularly egregious: the injector includes banner ads and fake search results that relegate all original page content beyond the browser's screen size. The same extension also hijacks a user's clicks and redirects the browser to product survey pages. "Uninstall"

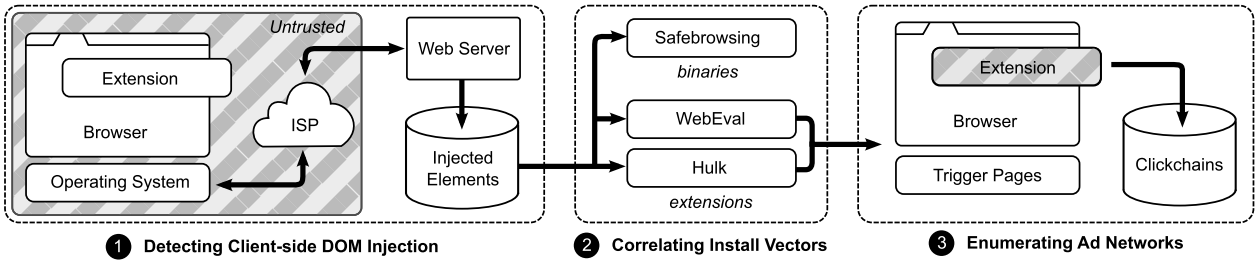


Figure 2: Workflow of our ad injection detection and analysis pipeline. We scan the client-side DOM of visitors to Google websites and report all injected elements (❶). We identify binaries and extensions that exhibit these behaviors (❷) and execute them in isolation to click on the resulting injected ads to enumerate advertisers and affiliates (❸).

options that sometimes accompany ads on the page instead redirect the user to install a multitude of bundled software packages. These practices highlight the often deceptive nature in which ad injectors operate.

C. Revenue Model

Ad injectors act like publishers in the traditional advertising ecosystem, pulling ads directly from advertising exchanges or affiliate programs (which we refer to as "intermediaries"). Relevant revenue models include *cost-per-click*, *cost-per-mille* (impression), and *cost-per-acquisition* (sharing profit from product sales). However, where traditional publishers have to generate popular content to attract users, ad injectors have complete control over all content that a client's browser renders. Consequently, injectors expose users to rogue advertisements as they browse; ad networks receive legitimate traffic; and the ad injectors, advertisers, and ad networks profit. However, what is unapparent to intermediaries throughout this process is the negative impact of ad injection on a user's browsing experience and the diversion of funds from webpage owners who have little recourse in the matter.

III. Methodology

We next turn to our system and data sources for studying the end-to-end ad injection ecosystem, outlined in Figure 2. We begin by ❶ scanning the client-side DOM of visitors to Google websites to identify the side-effects of ad injection; ❷ dynamically executing binaries and extensions in search of the same side-effects; and ❸ executing ad injectors in a contained environment while visiting numerous webpages to harvest advertisement clickchains and analyze the entities involved. For each of these components, we detail our design decisions, implementation, and any limitations or biases of our approach.

A. Detecting Client-side DOM Injection

Ad injectors rely on inserting rogue elements or modifying existing elements in a client's local rendering of HTML. We detect these artifacts by embedding a script in each served page that reports on the integrity of the client's DOM. Our script's payload contains a whitelist of domains and JavaScript handlers that we know *a priori* to appear in an untampered copy of the page. Once a pre-defined wait period elapses or when the browser signals a JavaScript page unload event, the payload scans the local DOM and identifies all `<a>`, `<script>`, and `<iframe>` elements on the page. If any of the elements identified violate the whitelist or have modified JavaScript events, we add them to a report returned to the web server. In the event we detect no DOM alterations, we return an empty report. This report also includes the URL of the page visited, the browser's user agent, and the IP address of the client (used only for geolocation purposes and client population estimates). We deployed our system on a select number of Google websites from June 1–September 30, 2014. The experiment targeted a random sample of Chrome, Firefox, and Internet Explorer desktop users for all operating systems and geographic regions. In total, we collected telemetry data on 102,562,842 page views.

We note that this technique detects *all* such DOM alterations, not just those caused by ad injectors. We rely on a post-report filtering phase to exclude browser toolbars, antivirus engines, or other programs that extend the behavior of pages. This filter leverages the fact that 89% of tampered views contain a rogue (non-whitelisted) script. We key in on which scripts inject ads—which we refer to as *injection libraries*—and in the process contextualize the other rogue URL and iframe elements that appear in our reports.

To simplify this process, we narrow our analysis to only the most popular scripts. Accordingly, we first normalize scripts to strip out any URL parameters. We then rank the scripts in order of the number of client DOMs they appear in. This ranking adheres to a Zipf-like distribution, shown in Figure 3. The single most popular script appears in 39% of reports and the top 100 in 74% of all tampered clients. The tail consist of over 19,315 scripts from 8,527 domains. Some of these top scripts are popular support libraries like jQuery used by benign extensions whereas others belong to ad injectors.

To filter out extraneous scripts, we manually reviewed the top 100 scripts and flagged them as injection libraries based on the script's content. In total we flagged 65 of the top 100 scripts as injection libraries. Of these, 31 actively cloaked against Google (e.g., network requests for the scripts returned a 404 error or empty DNS record to Google IP addresses or resolvers, but not to requests from independent vantage points). We offer a more detailed treatment of each script's behavior and the coverage this step provides in Section IV.

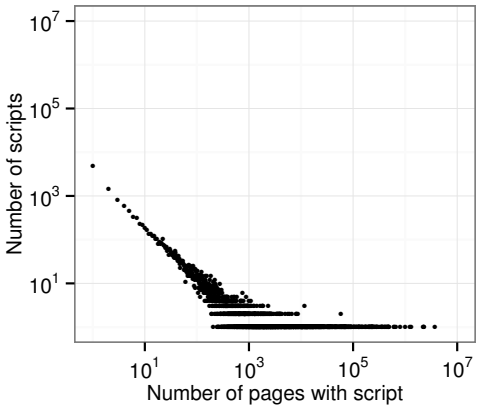


Figure 3: Zipf-like distribution of script popularity in tampered DOM clients. The top 10 and top 100 scripts appear in 56% and 74% of tampered DOMs respectively.

Design Decisions & Related Approaches: When we designed our client-side detection, we opted to whitelist rather than blacklist page elements as the former requires no knowledge of active threats and is more robust to changes in ad injection techniques over time. Our strategy is similar to "web tripwires" used to detect in-flight page changes [29]. We expand on the core idea of web tripwires and identify specific injected elements, rather than simply reporting a binary determination of whether tampering occurred.

Our system is also similar to Content Security Policies (CSP) supported by modern browsers [36], which rely on server-specified whitelists for preventing or reporting when a client's browser requests cross-origin scripts, iframes, or data (typically to prevent cross-site scripting attacks). We opted to rely on a JavaScript payload as it is simpler to deploy as an experiment, allows finer-grain reporting detail compared to CSP (e.g., page elements, not just cross-origin requests), and does not rely on browser compatibility beyond JavaScript. At the time of writing, CSP is only supported in Chrome, Firefox, and Safari; enforcement in Internet Explorer remains in development. A second motivation for JavaScript over CSP is that there is an existing incentive for extensions, binaries, and networks to tamper with CSP headers[2] due to enforcement policies that might otherwise prevent ad injection. Despite these shortcomings, we believe that CSP is a viable fallback for detecting ad injection in the absence of a more robust experimentation framework.

Ethics and Privacy: Our data collection technique is analogous to Content Security Policies, which modern browsers use to report client-side telemetry of page integrity to website operators. Prior to deploying our system, Google's internal privacy review board (similar to an IRB) vetted and approved our architecture and the data it collects. Part of the restrictions placed on our system include never analyzing data in non-aggregate form; never tracking or analyzing the behavior of individual users; never sharing raw data outside of Google; and setting a short lifetime on the data collected after which only aggregates could be stored. We also verified that our system is compatible for all clients that we collect data from (e.g., there is no degradation in the client's user experience due to crashing) and that our system *never* interferes with ad injectors or tampers with the client's DOM, but only provides a passive measure of ad injection in the wild.

[2]Web servers specify a CSP policy in HTTP headers for pages served to a client. Any intermediary with sufficient network control can strip or modify these values.

B. Identifying Distribution Vectors

We correlated the ad injectors we observed in client traffic with potential installation vectors, including extensions, binaries, and network tampering. To this end, we cast a wide net and dynamically analyzed over 1 million extensions and 25 million binaries in search of ad injectors. We used a secondary technique for identifying network-based injectors.

1) Browser Extensions: To analyze ad injection conducted via browser extensions, we relied on WebEval—Google's internal system for reviewing Chrome extensions which we describe below—and Hulk [18]—an independent system for detecting malicious Chrome extensions. We obtained extensions from three sources: crawling the web; extensions installed or side-loaded by binaries provided by Safe Browsing (described shortly); and all extensions in the Chrome Web Store. We used WebEval to evaluate all three sources—totaling over 1 million extensions created between March, 2011–October, 2014—while Hulk's analysis focused on 91,660 extensions it crawled from the Web Store.

We outline WebEval's analysis pipeline in Figure 4. An extension's evaluation consists of two phases: (1) a static scan of the extension's HTML, JavaScript, and manifest permissions; and (2) a dynamic evaluation, which loads a fresh copy of Chrome with the extension installed and subjects the extension to a barrage of behavioral suites. The full details of WebEval's architecture are beyond the scope of this paper, but we highlight the features relevant to our ad injection study.

Static Analysis: WebEval's static analysis module identifies the use of sensitive manifest permissions such as intercepting web requests (e.g., modifying incoming and outgoing network traffic to the browser), unrestricted access to cookies, or the ability to prevent uninstalling an extension. It also heuristically scans for code obfuscation, the inclusion of JavaScript `eval()`, and embedded URLs for remote resources (e.g., scripts, images). Static signals also include reputation data surrounding the extension developer, the age of the extension, the timeline of installs for the extension, and whether installs originate organically from the Web Store. Our install statistics are limited to aggregate counts (we have no details on which clients install an extension).

Dynamic Analysis: During dynamic analysis, WebEval launches a virtualized Windows environment and installs the extension under evaluation in an instrumented Chrome browser. The testing environment captures all Chrome API calls, DOM method calls, and network requests made by the

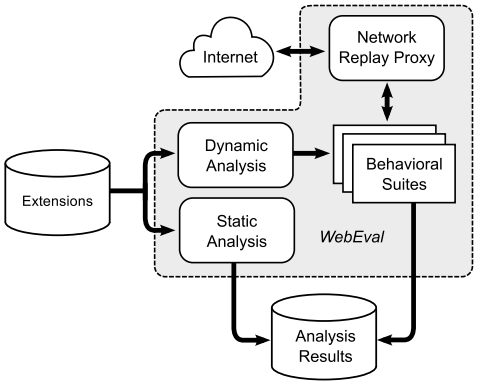


Figure 4: WebEval's analysis pipeline for scanning Chrome Web Store extensions.

browser during execution. WebEval then subjects the browser to behavioral suites that include querying Google Search and visiting dummy pages laden with advertisements to determine whether the extension alters the browser's DOM (e.g., injects new elements or replaces existing ads). WebEval specially constructs behavioral suites to replay network requests from a cached network trace previously produced by loading web pages in the same environment, but without the extension installed; any new network requests made by the extension are allowed to access the external Internet. This technique—similar to recording and replaying virtual machine system calls [13]—allows WebEval to identify all network requests made specifically by an extension rather than a web page. Furthermore, it prevents page dynamism from introducing false positives when detecting DOM-level alterations. As pages become stale over time WebEval periodically updates its test suites to keep up with content drift.

We combine WebEval's ad analysis suite and network traces with similar tests produced by Hulk. We then key in on the ad injection extensions that we also observe impacting clients in the wild. In particular, we manually construct a list of all of the top 65 injection libraries we observe correlated with ad injection on Google websites (discussed in Section III-A) and scan the network traces produced by WebEval and Hulk for these same scripts. If we observe network requests for injection libraries and the presence of rogue DOM elements, we label the extension as an ad injector. Of the extensions we analyze, we classify a total of 50,870 as ad injectors.

2) Binaries: We rely on Google's Safe Browsing infrastructure to dynamically scan hundreds of thousands of binaries daily, the details of which are previously described by Rajab et al [28]. Safe Browsing fetches binaries via crawls of the Internet as well as from payloads delivered by websites serving drive-by downloads in the wild [27]. As part of Safe Browsing's binary analysis, the system loads a virtualized environment and installs the binary under analysis. The system then launches Chrome and Internet Explorer and directs each browser to fetch a suite of websites which includes Google properties. The system logs all network traffic throughout this process, actively man-in-the-middling any HTTPS connections to introspect on packet contents. Similar to our extension analysis pipeline, we detect binary-based ad injectors by scanning the network logs related to visiting Google websites for outgoing requests to injection libraries. In total, Safe Browsing dynamically evaluated over 25 million unique binaries from February–October, 2014. Of these, we categorize 34,407 as ad injectors.

3) Network: We use the data produced by our client-side measurements to detect network-based ad injectors. Mechanically, these actors intercept page content in transit to either inject scripts or directly inject ad-based DOM elements. We detect the possibility of such tampering by comparing the fraction of tampered DOMs that were served over HTTP versus HTTPS. This approach assumes that a network attacker does not have a valid certificate to man-in-the-middle HTTPS connections (and that certificate pinning is not present to prevent this attack). This approach cannot determine which network element in particular is responsible for tampering with a client's connection.

C. Pinpointing Advertisers and Intermediaries

The ad injection ecosystem hinges on the willingness of advertisers to pay for traffic from injectors. This involves a tangled web of intermediaries that ultimately connect advertisers with injection libraries. We unravel this graph by automatically visiting *creatives* (e.g., ad text, images, and objects) served by ad injectors and enumerating all of the parties involved.

As a first step, we need to acquire creatives from ad injectors. One strategy would be to reverse engineer the ad request protocol for each of the ad injectors we encounter. While semi-automated approaches exist for extracting gadgets to replay malware C&C protocols [5], the security community relies on these techniques to overcome event-based triggers (e.g., timing, user interaction) that cause the malware to activate. In our case, we know the triggers that cause ad injectors to activate—it is as simple as visiting a web page.

To this end, we dynamically execute Chrome in a virtualized environment with an ad injector installed and then automatically visit a suite of pages that trigger ad insertion. This approach elides any complexities surrounding reversing the protocol of each injection library, executing JavaScript, priming cookies, or satisfying environment variables or parameters expected by the ad injector to operate. We select ad injectors from our feed of extensions, omitting binaries from the process. As we will discuss in Section V, extensions provide a better coverage of ad injectors in the wild.

1) Identifying Potential Trigger Pages: In order to generate a suite of *trigger pages* that reliably induce ad injection scripts to fetch rogue creatives, we manually scan the Alexa Top 100 in Chrome, cycling through 14 different ad injection extensions. This set of extensions provides non-overlapping coverage of the top injection libraries we observe in the wild, absent those not contacted by any of the extensions (or binaries) in our dataset. We specifically choose extensions

that fetch a single injection library as opposed to multiple libraries as not to conflate the websites targeted by individual libraries. In total, we visit 1,400 pages and manually identify the insertion or modification of ads.

Of the Alexa Top 100, we observe in-property injections into 75% of pages. These websites span search engines including *google.** and *yahoo.**; shopping pages including *amazon.**, *alibaba.com*, *ebay.com*, and *craigslist.org*; video sites including *youtube.com*, *dailymotion.com*, and *xvideo.com* (adult); and other popular news and media sites. Based on the most popular websites targeted by multiple injection libraries, we elect to use *google.com*, *amazon.com*, and *walmart.com* as trigger pages for inducing ad injector behaviors. These websites are convenient as we can easily generate a set of 100 representative queries for each site. For Google, we obtain a list of the top 100 search queries that contain advertisements. For Amazon and Walmart, each site publishes a leader board of the top 100 best selling products [2], [35]. While these 300 queries will not provide complete coverage of the long tail of creatives accessible to ad injectors, they offer a starting point for understanding the advertising relationships underpinning injectors.

2) Enumerating Intermediaries: We select a random sample of ad injection extensions and successively visit each of our 300 trigger pages with each extension installed on a client outside Google's IP space to avoid cloaking. We identify creatives inserted into each property by extending our client-side DOM measurement technique. In particular, after each page finishes loading and a predefined wait expires, we inject a script that scans all `<a>`, `<script>`, `<iframe>`, `<object>`, and `<div>` elements. For each property we develop a comprehensive whitelist of JavaScript events, domains, and div classes and IDs that appear independent of ad injectors, also accounting for page dynamism. We consider any element that violates this whitelist as a potential ad to click on. Unlike in our client-side measurement, we are able to access nested cross-origin content by running Chrome with same-origin protections disabled. As such we can observe all injected content, not just that in the parent frame. This is critical as many intermediaries syndicate content, with syndication represented by layered iframes where the lowest layer contains the creative.

We finally uncover the ad revenue chain underpinning a creative by clicking on one injected ad per page if an ad is present. During this process we monitor the creation of new tabs and examine all network traffic produced by Chrome until the browser fetches the final landing page. We natively generate clicks rather than via JavaScript in order to support Flash ads. We repeat this process indefinitely: fetching one extension from a circular task queue, running it against a trigger page, and then tearing down the browser session after each successful click before moving on to the next trigger page. In total, we generate 114,999 ad revenue chains from 398 distinct extensions. Of these, 62,237 were from ads injected on Amazon, 37,718 from ads on Google, and 15,044 ads from Walmart.

Dataset	Source	Sample Size
Client DOM reports	*Client-side scan via Google properties*	102,562,842
Unique extensions	*Dynamic evaluation via WebEval, Hulk*	$>$ 1,000,000
Ad injection extensions		50,870
Unique binaries	*Dynamic evaluation via Safe Browsing*	$>$ 25,000,000
Ad injection binaries		34,407
Ad revenue chains	*Trigger and click analysis on Google, Amazon, Walmart*	114,999

Table I: Summary of datasets produced or consumed by our analysis pipeline.

D. Dataset Summary

We summarize the datasets produced and consumed by each stage of our pipeline in Table I. In total, we collected 102,562,842 client-side DOM reports through which we identified the 65 most popular ad injectors. We then statically analyzed and dynamically executed over 1 million extensions and 25 million binaries in search of these ad injectors, identifying over 50,870 culpable extensions and 34,407 binaries. Finally, we executed a sample of the ad injection extensions to uncover 114,999 clickchains to pinpoint intermediaries involved in supporting ad injection. The remainder of this work focuses on insights gleaned from each of these datasets.

E. Limitations

1) Client DOM reports: Our design has a number of limitations imposed by vantage points, browser security models, size constraints, and operating in a hostile environment, which may bias our technique towards catching specific ad injectors. While we believe our system can yield meaningful insights, we are nevertheless cautious to draw conclusions about all ad injectors in the wild.

Visibility into the Internet: Our experiment's vantage points are limited to Google websites. While we confirmed the ad injectors we identify target many of the Alexa Top 100, and thus generalize, there may be other ad injectors that tamper solely with non-Google sites. We believe it is critical for other websites to repeat our study to capture the full extent of ad injection in the wild.

Same-Origin Restrictions: Browsers enforce a same-origin policy that restricts our client-side analysis from accessing content outside the current page's origin. Consequently, nested content in cross-origin frames is invisible to our scanning beyond the outermost container. As such, our reports may contain a subset of all `<script>`, `<iframe>`, or `<a>` elements that browsers render on a page and a narrower perspective on the ad injectors responsible.

Detecting Modified Events: Our whitelist consists of both domains as well as JavaScript events associated with page elements. While this JavaScript whitelist accurately captures `onClick` and other explicit `onMouse` handlers, events added to elements via the `addEventListener` method are invisible

to our scanner. This limitation stems directly from the lack of JavaScript support for enumerating event handlers without debug privileges or intercepting calls to the `addEventListener` method (which might otherwise result in race conditions). As such, some tampered elements may go undetected by our system.

Whitelist Specificity: We elect for a domain whitelist rather than a full URI whitelist for both space constraints on the size of our script payload as well as for contending with highly dynamic page content where we cannot easily generate a comprehensive *a priori* whitelist. Consequently, this approach will overlook injected elements that direct to whitelisted domains (e.g., new DoubleClick ads on pages that previously served DoubleClick ads) or when ad injectors tamper with URL parameters (in particular affiliate IDs). Again, this may result in some injected or tampered elements going undetected.

Report Tampering: Our final concern relates to operating in a hostile environment where we cannot trust a client's browser, operating system, or network connection to not interfere with our scanning or reporting. While we believe this is unlikely as we deployed our detection scheme over a short period without any external signs of enforcement (e.g., we *never* interfere with ad injectors), we rely on an encrypted delivery and reporting mechanism that is uniquely keyed per client. (The details of this mechanism are beyond the scope of this paper.) We note that such protections will not prevent a determined adversary, but the delivery mechanism provides a degree of application-level protection against rudimentary man-in-the-middle attacks.

2) Binary & Extension Coverage: Our coverage of binaries and extensions is limited to software that Safe Browsing, WebEval, and Hulk encounter. For each of these systems, there is a possibility of ad injection software cloaking against dynamic analysis or simply not triggering for the tests we execute. We provide an estimate of our coverage of software-based ad injection later in Section V.

3) Advertiser & Intermediary Coverage: Our decision to use 300 Google, Amazon, and Walmart search and product results limits the scope of creatives that ad injectors may serve. This restriction in part arises due to the tedious, sometimes manual effort required to construct stable whitelists for highly dynamic websites. Consequently, we will miss creatives tailored specifically to the long tail of products, news sites, or pornographic content. As such, our clickchain analysis only captures a subset of possible advertising relationships. Furthermore, we only collect a relatively small sample of clickchain data to minimize the ad revenue impact we have on advertisers buying traffic from ad injectors. We make no effort to obfuscate our IP address or evade any automated clicking protections used by advertisers.

IV. Ad Injection in the Wild

Ad injection has entrenched itself as a cross-browser monetization platform that impacts tens of millions of users around the globe. We find that many ad injectors are organized as affiliate programs. In this model, third parties are responsible for obtaining installs while ad injection libraries provided by the affiliate program manage advertisement selection. We provide a perspective of the most popular injection libraries, the number of affiliates each program attracts, each affiliate's user base, and how programs compete.

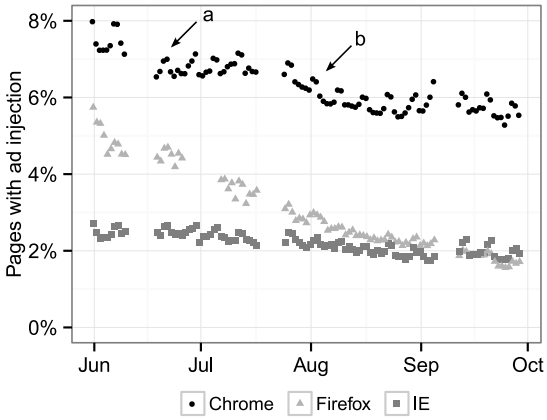


Figure 5: Prevalence of ad injection in client-side DOMs, broken down by browser. Dates are 2014. Gaps in coverage result from Google deploying our client-side experiment for short periods before electing for continuous deployment. See text for discussion of the a and b changes.

A. Prevalence of Ad Injection

Of the client-side reports we collect, we find that 5,339,913 (5.2%) contain evidence of ad injection. If we consider IP addresses as unique client identifiers, we find that injections impact a daily average of 5.5% of unique daily IPs.[3] Figure 5 contains a detailed breakdown of injection levels over time per browser.

Chrome, the most popular browser in our dataset, is the most commonly affected platform (5–8% of page views). During our measurement, we observed two declines in impacted Chrome views around mid-June and late-July. We believe the first drop (a) is related to the Chrome Web Store lockdown where Chrome prevented side loading extensions on Windows that were not also in the Chrome Web Store [20]. The second drop (b) correlates with the enforcement of an updated Chrome Web Store policy on "single purpose" extensions [19], e.g., explicitly prohibiting marketing an extension as a game and then also surreptitiously performing ad injection on search results. We note that ad injection, when properly disclosed to users, is not explicitly prohibited by the Chrome Web Store. While both policies appear to impact ad injection levels, there remains a substantial volume of ad injection on Chrome.

For Internet Explorer, ad injection levels have remained steady, impacting around 2% of page views. Conversely, nearly 6% of Firefox views exhibited signs of ad injection, though

[3]We note that dynamic leases on IPs and NATing inject a certain noise into this user estimate. Our data collection policy precludes mechanisms to uniquely identify clients (e.g., cookies) needed to correct for this uncertainty.

Rank	Ad Injection Library	Impacted Views	Popularity
1	superfish.com	3,751,167	3.92%
2	api.jollywallet.com	2,292,685	2.40%
3	visadd.com	1,337,099	1.40%
4	intext.nav-links.com	1,231,504	1.29%
5	{crdrdpjs, rvzrjs, ...}.info	665,505	0.70%
6	ads.tfxiq.com	472,745	0.49%
7	noproblemppc.com	423,682	0.44%
8	clkmon.com	358,863	0.38%
9	datafastguru.info	294,261	0.31%
10	easyinline.com	206,157	0.22%
11	donation-tools.org	147,368	0.15%
12	rjs.mzcdn.com	127,849	0.13%
13	apimegabrowsebiz-a.akamai...	105,844	0.11%
14	fcdn.smileyswelove.com	94,665	0.10%
15	apibrowsemarknet-a.akamai...	86,722	0.09%
16	apisurftasticnet-a.akamai...	64,785	0.07%
17	jscripts.org	63,345	0.07%
18	apiwisenwizardne-a.akamai...	60,656	0.06%
19	cdn.taboola.com	60,213	0.06%
20	savingsslider-a.akamai...	59,839	0.06%
21	apimyfindrightco-a.akamai...	59,344	0.06%
22	jsutils.net	45,113	0.05%
23	static.dreamsadnetwork.com	24,780	0.03%
24	gyr.mappingsection.net	22,490	0.02%
25	cluster.adultadworld.com	20,846	0.02%

Table II: List of the top 25 ad injection libraries we observe in client traffic. We combine 224 algorithmically generated domains such as *crdrdpjs.info* and *rvzrjs.info* that all host the same library into a single grouping for clarity.

this has steadily declined since we began our measurement. We are unaware of any action taken by Firefox against extensions or hardening the browser against DOM tampering that would explain this decline.[4]

B. Most Popular Injectors

A select few injection libraries dominate the ad injection ecosystem, detailed in Table II. We find that each injector is diligent in universally supporting browsers; of the top 25 injectors, all target Chrome, 24 target Firefox, and 23 target Internet Explorer. Far and away, *superfish.com* is the most popular program and appears in 3.9% of Google views.[5] Its parent company markets itself as an image similarity search, with the injection library offering alternative shopping suggestions in the form of ads on shopping or search properties a client visits. The second most popular program is *jollywallet.com* (2.4%), which overwrites affiliate parameters for URLs on shopping sites to monetize *cost-per-acquisition* revenue sharing models without actually driving traffic to the shopping partner.[6] While this does not fall directly into our definition of ad injection, we nevertheless include it in our subsequent analysis as it often co-occurs with other injection libraries. It is the only affiliate re-writer we encounter in the top 100 injection libraries. We provide a more detailed treatment of each the top injection libraries later in Section VII.

[4]We note that the decline in Firefox injection levels is independent of the browser's update release cycle. We observed no discernible drop in injection levels between version changes.

[5]Subsequent to our study, it emerged that a popular laptop vendor, Lenovo, had pre-installed Superfish on some of their devices [23]. Our dataset predates this event, though given the range of installation vectors we observe, we believe OEM packaging is secondary to installs originating from download bundles and malware.

[6]We cannot directly detect affiliate modifications made by Jollywallet with our current client-side scan. Instead, we detect the injection library that performs affiliate re-writing to measure impacted clients.

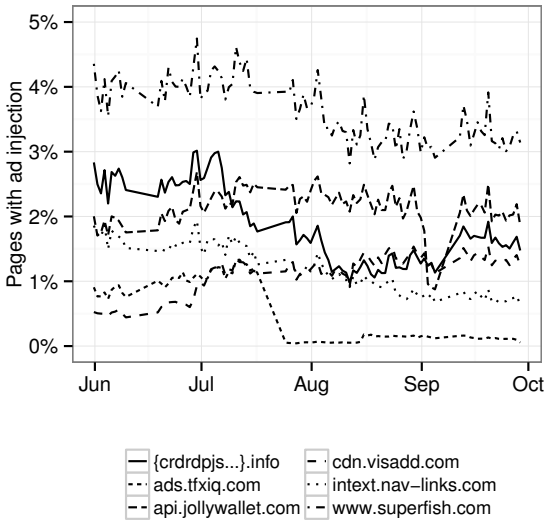


Figure 6: Prevalence of the top ad injection libraries in Google pages over time.

Each injection library's prevalence is constantly in flux, as outlined in Figure 6. We observe a substantial drop in *superfish.com* injections in early August that correlates with the Chrome Web Store removing deceptive extensions [19]. Conversely, *visadd.com* has experienced a steady growth from roughly 0.5% of traffic to about 1.4% at the end of our measurement. This same period shows a substantial decline for *ads.tfxiq.com* and *intext.nav-links.com*. The process that drives the adoption or deactivation of injection libraries is unclear, but the most popular programs appear to actively cultivate a steady user base to drive revenue.

C. Affiliate Structure of Ad Injectors

When we manually reviewed the top injection libraries we found that many were in fact organized as affiliate programs. In this model, *affiliates* embed the injection library provided by an *affiliate program* into a client's browsing session—using whatever mechanism available—which then handles ad selection. The affiliate program in turn pays affiliates for the volume of traffic or installs they generate. For instance, *superfish.com* recently appeared at Affiliate Summit East (a popular recruiting fair for affiliates), with one of the company's employees stating "Let's meet up to monetize your toolbar or add on" [4].

We manually reverse engineered the set of URL parameters associated with affiliate tracking for each of the top ad injection affiliate programs and then scanned every record in our dataset to pull out affiliate data. We provide a breakdown of the total unique affiliates we identify in Table III. We find that affiliate market shares of each affiliate program's total traffic follow a long tail distribution. The top 10% of affiliates control 47–96% of each affiliate programs' market share, while the

Ad Injection Library	# Affiliates	Top Affiliate	Top Share
superfish.com	494	Crossrider	44%
api.jollywallet.com	114,486	Shopperpro	8%
visadd.com	885	Iwebar	7%
intext.nav-links.com	96	Crossrider	42%
{crdrdpjs, rvzrjs, ...}.info	613	TornTV V9.0	7%
ads.tfxiq.com	479	Sense	8%
clkmon.com	521	Plus-HD-9.4	6%
datafastguru.info	57,983	Browser Shop	10%
easyinline.com	155	Netcrawl	12%

Table III: Affiliate structure of the top ad injectors, the total number of unique affiliates identified in client-side DOMs, the largest affiliate by market share, and the market share the they control.

top 50% control 70–99%. This is particularly apparent for *superfish.com* and *intext.nav-links.com* where the top affiliate—Crossrider—controls 42–44% of the market share. Crossrider is a mobile, desktop, and extension development platform that enables drop-in monetization via major ad injectors. Crossrider provides its affiliate ID to ad injectors while separately tracking kick-backs to developers. The other top affiliates listed in Table III are all cross-browser extensions and plugins that impact Chrome, Firefox, and Internet Explorer.

D. Co-occuring Ad Injectors

The affiliate nature of ad injection creates an incentive for software developers and network operators to laden impacted clients with ads supplied from as many affiliate programs as possible. We observe that 50% of tampered clients fetch at least two injection libraries per page load and 30% of clients at least four. To better understand the practice of bundling multiple injection libraries per client, we calculate the Jaccard similarity coefficient between the pages impacted by distinct injection libraries. We show our results in Figure 7. The most popular program, *superfish.com*, appears 49% of the time with *jollywallet.com*, 33% with *visadd.com*, and 25% with *intext.nav-links.com*. A similar pattern emerges between other affiliate programs.

This practice maximizes the profit that affiliates earn per client. As we discuss in Section VII, each injection library has varying advertising relationships that may or may not monetize certain traffic (e.g., pornographic content). Similarly, many of the top injection libraries include a blacklist of properties they abort tampering with, not all of which are consistent. Affiliates overcome these gaps in coverage by contacting as many affiliate programs as possible. We support this conclusion further in Section V where we observe binaries and extensions simultaneously installing multiple injection libraries.

E. Understanding the Ad Injector User Base

Ad injection impacts users located around the globe independent of the operating system they use. We find that injectors tamper with 5.11% of pages served to Windows machines and 3.43% of page views on Mac OSX. This indicates that ad injection is not a Windows binary problem, but rather broadly impacts diverse operating systems and browsers.

Figure 8 shows a breakdown of ad injection levels per region, as determined by the geolocation of clients. We note

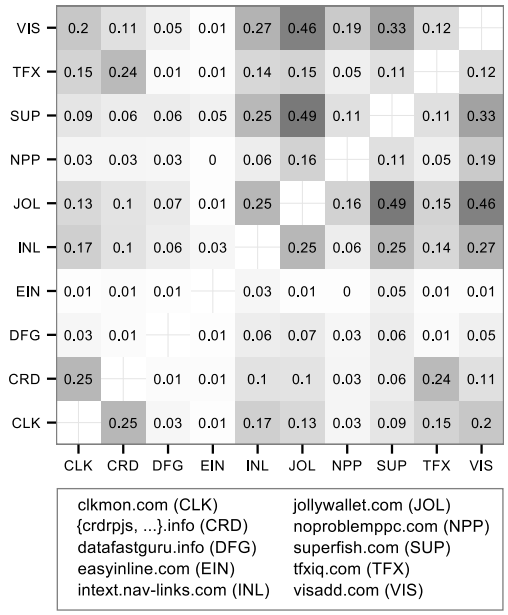


Figure 7: Overlap between ad injection libraries, as measured by the Jaccard similarity between the sets of impacted clients.

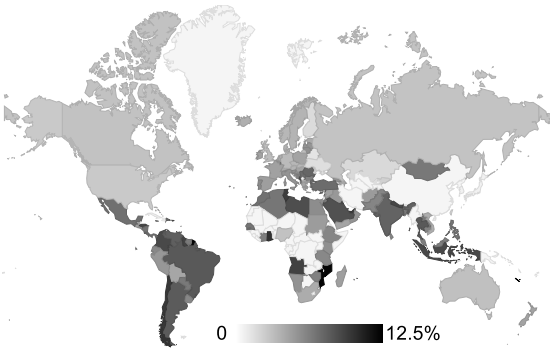


Figure 8: Impact of ad injection around the globe. South America, South Asia, and South East Asia have the highest concentration of injection.

that regions in white either block access to Google (e.g., China) or are omitted due to a limited sample size (e.g., Central Africa). We observe that South America, South Asia, and South East Asia are the most frequently affected regions (8% of views). Conversely, ad injection impacts only 2.6% of North American views, 3–4.5% of European views, and fewer than 0.6% of Japanese and South Korean views.

F. Long Tail of Ad Injectors

Our methodology for detecting ad injectors concentrates on the top 65 scripts used by ad injectors. As we discussed in Section III, in practice there are 19,315 rogue scripts appearing in DOM content. We place an upper bound on the fraction of views impacted by ad injectors in this long tail by calculating the fraction of all page views that include *any* script outside of the 35 in the top 100 we determined to be benign (Section III-A). In the worst case, if all rogue scripts relate to injection libraries, it would at most impact an additional 3% of Google page views irrespective of the browser involved.

V. Distribution & Injection Techniques

Injected ads arrive on a client's machine through multiple vectors: binaries, extensions, and even network-level tampering. We correlate each of these distribution channels with the most prominent ad injectors; delve into the technical mechanisms that extensions and binaries use to modify client DOM content; estimate the number of Chrome users impacted specifically by ad injection extensions; and finally estimate our coverage of ad injection software.

A. Ad Injector Distribution Channels

In total, WebEval, Hulk, and Safe Browsing identify 50,870 extensions and 34,407 binaries that perform ad injection. New ad injection software is constantly emerging. Figure 9 shows a log-scale timeline of new extension and binary variants throughout October 2013–October 2014. In April 2013, over 1,000 new ad injection extensions were authored. Since that period, new extensions have steadily declined to roughly 10 per day. Conversely, the arrival of new binaries defies any trend. The largest peak of over 10,000 new Windows installers occurred around July 2014, with numbers dropping to roughly 100 per day in October 2014. Of the extensions we identify, only 10% were ever present in the Chrome Web Store. The remaining 90% originate from binaries or websites providing off-market extensions which can no longer be installed on Windows.[7]

We provide a breakdown of the most popular ad injection libraries used by software in Table IV. We find that 49,127 (96%) different extension variants and 33,486 (97%) distinct binaries contact *superfish.com*, the first sample of which appeared back in September 2012. As we observed in client traffic, the authors of these programs frequently bundle multiple injection libraries at a time. Of extensions, 50% contact at least two injection libraries, while 30% contact at least five. Similarly, 80% of binaries contact at least four injection libraries. We were unable to identify any extensions or binaries that contacted *easyinline.com*, likely due to incomplete coverage of software in the wild.

B. Dissecting Ad Injection Techniques

1) Extensions: Ad injection extensions rely on the Chrome permission model [3] to request access to DOM content and privileged browser resources. We statically analyze the manifests embedded in each extension to determine which permissions an extension requests, the details of which appear in Table V. Permissions consist of a scope as well as a resource. Of the extensions in our dataset, 100% scope their privileges to *every* page a client visits, indicated by http(s)://*/* or the <all_url> scope. This behavior is also typical of malicious browser extensions [18]. Within this broad scope, 69% of extensions can potentially prevent uninstallation

[7]We note that extension side-loading can still occur if the extension is simultaneously in the Chrome Web Store. The Chrome Web Store team, however, recently implemented a policy to take action on any extensions that benefit from installation tactics that deceive the user or otherwise violate the Unwanted Software Policy.

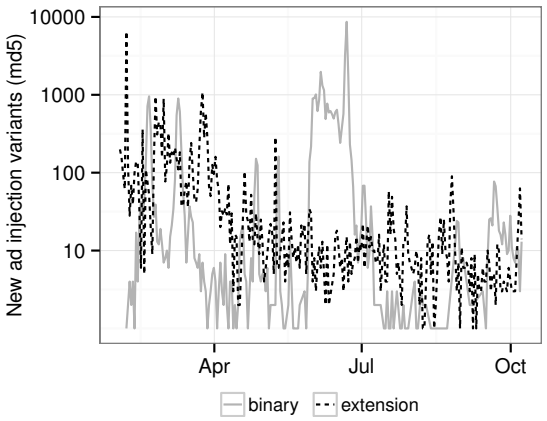


Figure 9: Arrival rate of new extension and binary variants that perform ad injection from October 2013–October 2014.

Ad Injection Library	Extensions	Binaries	Earliest Trace
superfish.com	49,127	33,486	Sep 09, 2012
api.jollywallet.com	19,259	28,557	Jul 12, 2013
visadd.com	11,843	13,763	Jan 29, 2014
intext.nav-links.com	17,007	4,881	Jul 26, 2013
{crdrdpjs, rvzrjs, ...}.info	16,381	28,574	Mar 04, 2013
ads.tfxiq.com	248	1	Mar 16, 2014
noproblemppc.com	18,228	28,972	Sep 15, 2012
clkmon.com	27	1	Nov 20, 2013
datafastguru.info	4,221	6	Dec 16, 2013
easyinline.com	0	0	–
Total	50,870	34,407	–

Table IV: Breakdown of extensions and binaries performing ad injection. We obtain tens of thousands of samples for 9 of the top 10 injectors. We denote the earliest date we identify either an extension or binary contacting an injection library.

by auto-closing tabs directing to `chrome://extensions`; 54% can access a client's cookie for any property; and another 52% can monitor the installation of other extensions.

From this list of permissions, we discern that ad injection rarely occurs via in-browser network interception denoted by the webRequest permission (5%) or plugin access to install new DLLs (2%). Instead, the extensions rely on content scripts, which are JavaScript files loaded into a page's context after the browser renders a document. Independent of the delivery mechanism, we find that 100% of extensions request to modify every HTTP and HTTPS page a client visits. With these privileges, ad injectors add new `<script>` elements that fetch remote copies of the most popular ad injection libraries into every DOM a client renders. These scripts in turn supply the logic for adding or replacing advertisements. Consequently, ad injectors can tamper with *any* page a client visits.

Extensions performing ad injection have a range of other behaviors. WebEval flagged 24% for spamming a user's social network and 11% for hijacking a user's search queries. Independent of our ad injection analysis, WebEval flagged 38% as malware. These results highlight the tangled nature of ad injection affiliate programs. Extension authors can bundle ad injection with any number of other monetization strategies,

Permission	Popularity	Description
https://*/*	88%	*Access all https pages*
http://*/*	88%	*Access all http pages*
<all_urls>	12%	*Access all pages, including ftp, data*
tabs	69%	*Create or modify tabs*
cookies	54%	*Access cookies for permitted sites*
management	52%	*Control removal of extensions*
webRequest	5%	*Intercept network requests*
webNavigation	3%	*Notifications for when pages change*
plugin	2%	*Install DLLs or other binary files*

Table V: Sensitive permissions requested by extensions performing ad injection. Injectors can manipulate any page a client visits, modify cookies, and prevent uninstallation.

including those traditionally relied on by the malware ecosystem.

2) Binaries: We observe two predominant strategies for tampering with DOM content via binaries: intercepting network requests and side-loading an extension. Of the binaries we detect as ad injectors, 86% modify the Window's registry key for `HKCU\software\microsoft\windows\currentversion\internet\proxyserver` to install a proxy. Without additional support from DLL injection or installing a new certificate, this proxy cannot tamper with HTTPS connections. Regretfully, we cannot discern such granular information for our current dataset. The remaining 14% of binaries silently install an extension by modifying the user's browser profile stored in `google\chrome\userdata\default\preferences`.

This ratio is skewed in part by multiple binary versions distributed by the same author. If we de-duplicate binaries based on the website hosting the binary, we observe 615 distinct domains in the last year. Of these, 74% serve binaries that install proxies, while 51% serve binaries that side-load extensions. As such, we are cautious in drawing conclusions on whether extension side-loading is secondary to installing proxies in the wild.

Independent of these behavioral signals, we examine the labels returned by VirusTotal and Safe Browsing. We find that AVG flagged 62.5% of binaries for search engine hijacking, while Sophos and Kaspersky flagged 46% and 39% of binaries respectively as major adware distributors. Not all binaries restrict their actions to greyware. Safe Browsing flagged 17% of ad injection binaries for also installing malware. Like extensions, our results show that ad injectors are often bundled with a multitude of unwanted and even malicious software.

3) Networks: We have no mechanism to attribute in-transit ad injection to the ISPs or the network routes involved. However, we can compare the discrepancy between the level of ad injection over HTTP versus HTTPS from our client DOM reports in Section IV. We observe that 4.5% of Google properties served to users over HTTPS contain traces of ad injection, compared to 6.1% of HTTP connections. This represents a 35% increase for unprotected traffic. Two possible explanations exist for this discrepancy. Either binaries restrict tampering to HTTP connections because they lack the technical components required to hook the network stack after libraries remove encryption—or, alternatively, there is a large amount of in-transit tampering.

C. Estimating Ad Injection Caused by Extensions

The Chrome Web Store tracks two metrics surrounding extensions: (1) an extension's *active install base* as determined from update requests sent by Chrome clients (even for extensions not in the official Web Store); and (2) an extension's *organic installs*—the total number of clients that ever installed the extension directly from the Chrome Web Store or via trusted UI elements that refer to the Web Store. We refer to all active install bases with no organic origin as inorganic installs. These can occur from binaries side loading extensions or users installing extensions from third-party websites (now defunct with the Chrome Web Store lockdown where all extensions must be in the official Web Store). As these install values are constantly in flux, we measure both as the maximum value over the lifetime of an extension.

Figure 10 provides a log-scale CDF of the historical maximum user base for every extension (many now defunct). We find that 50% of ad injectors never acquired more than 10 active users, while the largest user base for a single extension totaled over 10.7 million users. If we consider the user base of each extension to be unique, then a total of *103 million* Chrome users at one point had an ad injector installed. Some of the most successful ad injection extensions are organic (1.8% of total extensions); combined, they total 11.3 million organic installs. The remaining 98.2% of extensions are fueled by inorganic installs. Our results show that extensions play a major role in the ad injection ecosystem, but that a secondary delivery mechanism such as a binary is necessary for their installation.

D. Taking Action on Live Extensions

If we restrict ourselves to extensions that are currently active (e.g., not taken down from the Chrome Web Store), there are 249 ad injection extensions in our dataset impacting a combined 25 million users. We alerted the Chrome Web Store to these extensions in the event any violated the Chrome Web Store policy. Of the extensions, the Chrome Web Store classified 192 as deceptive in nature (e.g., violating the Single Purpose Policy [19]) that affected 14 million users; 6 were ad injectors that did not violate the Web Store policies and affected 28K users; 16 developers had since removed the ad injection component of their extension; and 35 belonged to developers now suspended for other reasons. Extensions removed for Single Purpose Policy violations are *not* immediately uninstalled from a user's machine.[8] As such, we observe no immediate impact from the Web Store's actions on our client-side measurement of ad injection levels.

[8]Client-side extensions that violate Single Purpose Policies may be disabled on users' machines if a developer fails to make her extension policy-compliant within a certain time window.

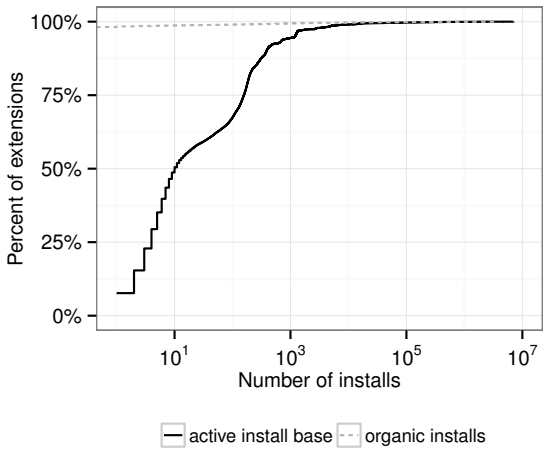


Figure 10: CDF of maximum active install base and organic installs for all ad injection extensions. Only 1.8% of ad injection extensions ever acquire organic installs.

Ad Injection Library	Extension Affiliates	Extension Coverage	Binary Affliates	Binary Coverage
superfish.com	149	88%	21	64%
api.jollywallet.com	127	30%	3	11%
visadd.com	1	0%	0	0%
intext.nav-links.com	18	65%	6	19%
{crdrdpjs, rvzrjs, ...}.info	149	36%	17	0%
ads.tfxiq.com	17	13%	1	7%
clkmon.com	0	0%	1	5%
datafastguru.info	16	3%	0	0%

Table VI: Unique affiliates we detect in dynamic traces of extensions and binaries, along with the volume of client DOMs that contain the same affiliate IDs. We obtain more comprehensive coverage of affiliates in client traffic than via our synthetic execution environment.

E. Estimating Coverage of Ad Injection Software

While we cast a wide net for ad injection software, we bias our understanding of the ecosystem to the input sources used by WebEval, Hulk, and Safe Browsing. We estimate our overall coverage by comparing the affiliate parameters that appear in extensions and binaries against those appearing in client DOMs analyzed in Section IV. We show our results in Table VI. Our coverage of different injectors varies heavily per affiliate program. Of *superfish.com* affiliates, we identify 149 IDs in our dynamic extension traces which correlate with the same IDs in 88% of client DOMs. The same analysis for binaries yields 21 affiliates accounting for 64% of client DOMs. The limited affiliates found for *visadd.com* and *clkmon.com* may result from insufficient browser interaction to elicit network traces with affiliate IDs. Our results indicate that we have a substantial sample of ad injection software, though we do not have complete coverage of all affiliate distribution techniques in the wild. Between extensions and binaries, our extensions dataset provides the best coverage of affiliates and ad injection programs.

VI. Identifying Advertisers and Intermediaries

Money enters the ad injection ecosystem through a tangled web of advertisers and intermediaries. We explore ads injected on Google, Amazon, and Walmart and identify over 3,000 websites that unwittingly purchase traffic from injectors. As we show, advertisers rarely have insight into the provenance of traffic; they only observe parameters tied to the last hop (e.g., HTTP redirect with a referrer) of the full clickchain. Accordingly, we illuminate the full chain of ad relationships that underpin the injection ecosystem and in the process highlight the intermediaries who can have the greatest impact on blocking deceptively sourced traffic.

Impacted Property	Tampered Trigger Pages	Click Chains	Ad Injection Libraries	Extension Coverage
amazon.com	90	62,237	27	86%
google.com	96	37,718	13	80%
walmart.com	71	15,044	25	91%

Table VII: Breakdown of the revenue chains we collect from ads injected into Amazon, Google, and Walmart and the distinct queries ads were sourced from.

A. Ads Injected on Amazon, Google, & Walmart

We aggregate a total of 114,999 revenue chains from ads injected by a sample of 398 extensions into Google, Amazon, and Walmart. These extensions provide overlapping coverage of all of the top injection libraries. A detailed breakdown of our dataset appears in Table VII. Of the 300 trigger pages we visit, 86% successfully induce an injected ad. We observe an average of 5 iframe ads, 110 divs, 0.4 flash ads, and 15 ad URLs per page. We randomly click one of these elements and collect the resulting revenue chains to enable studying the ad ecosystem. These revenue chains consist of every hand-off that occurs between Amazon, Google, or Walmart until the advertiser's landing page.

We provide a sample of a real revenue chain in Figure 11. When we visit *google.com* and query for Android, the injection library for Superfish triggers (❶) and fetches a list of advertisements to embed (❷). We randomly click the BestBuy offer for $39.99, kicking off a redirect chain through multiple intermediaries (❹). We finally arrive at the advertiser, BestBuy (❺). Worth noting, ad injectors cheekily view this as yet another opportunity for profit. In our example, the injection libraries in turn laden *bestbuy.com* with a multitude of rogue advertisements despite just having been paid by BestBuy to deliver traffic. We rely on this example as a reference for terminology used throughout the remainder of this section.

Table VIII shows a breakdown of the top 10 injected ad domains that start ad revenue chains. These parties are the primary source of the deceptively sourced traffic that pollutes the ad ecosystem. As we discussed in Section V, a single extension will generally maximize its profit by contacting two to four injection libraries. Our current infrastructure cannot determine the exact injection library responsible for inserting a specific ad. As such, we refer to injected ads by the domain of the first hop in the revenue chain, rather than the name of the injection library previously used throughout the paper. In some cases such as *superfish.com* these are the same. Conversely, *dealply.com* represents the amalgam of all the {*crdrpjs, ...*}.*info* injectors. Noticeably absent is

Figure 11: Real example of a clickchain produced from an ad injected on Google that ends at BestBuy (which is then laden with injected ads).

Injected Ad Domain	Click Samples	Avg Hops	Advertising Domains	Extension Coverage
superfish.com	63,891	3.6	891	60%
dealply.com	20,209	9.2	526	29%
datafastguru.info	3,899	4.9	407	25%
display-trk.com	3,353	12.7	196	27%
tfxiq.com	3,091	1.8	58	12%
pangora.com	1,814	2.9	204	5%
xingcloud.com	1,116	1.0	4	<1%
shoppingate.info	994	4.9	232	2%
linkfeed.org	822	6.3	267	12%
bestyoutubedownloader.com	688	1.0	1	3%
Other	15,122	7.48	1,497	100%

Table VIII: Top 10 injected ad domains we observe as the first hop in click revenue chains and the number of associated advertisers receiving traffic.

jolleywallet.com; this is because the program relies on affiliate referrals that do not exist on our trigger websites.

The top 10 injected ad domains contribute 87% of revenue chains followed by a long tail of over 15,112 distinct ad domains. We find *superfish.com* is the most popular source of traffic in our revenue chains, in part because 60% of the extensions we execute belong to its affiliate program. It is followed in popularity by *dealply.com* and *datafastguru.com*, which are contacted by 29% and 25% of extensions in our synthetic environment. We caution that our dataset is biased towards these top injected ad domains due their popularity among the extensions we dynamically execute. As such, we examine each injector on an individual basis rather than in aggregate.

Two striking results emerge from our results. First, hundreds of advertisers (i.e., landing pages) receive traffic from injection libraries, even for our limited set of 300 product search queries. We cannot illuminate what volume of ad traffic this represents in the wild. Secondly, the owners of injection libraries are closely connected to the ad ecosystem and advertisers. It takes only 3.9 hops (e.g., intermediaries) before traffic sourced by *superfish.com* reaches advertisers and 4.9 hops for *datafastguru.com*. Only *display-trk.com* is far removed at 12.7 hops away, indicating injected traffic requires a more complicated path before it reaches advertisers. We also observe landing pages receiving search traffic (*xingcloud.com*) and installs (*bestyoutubedownloader.com*) directly from injectors, indicating some immediate relationship exists as well.

B. Advertisers and Intermediates For Top Injectors

Given the sheer diversity of revenue paths available to injection libraries, we narrow our analysis towards the advertisers and intermediaries impacted by the top 3 injected ad domains in our revenue chains (and popular in user traffic). These contribute 77% of all injected ads from 81% of extensions.

Traffic Entry Points into the Ad Ecosystem: Ad injection traffic enters the ad ecosystem through a small number of paths, detailed in Table IX. The intermediaries involved in sourcing ad injection traffic are remarkably similar irrespective of the injector involved. We find that *superfish.com* funnels all of its traffic through three shopping programs: DealTime, PriceGrabber, and BizRate (also known as ShopZilla). BizRate is also used by *dealply.com* (67%) and *datafastguru.com* (43%). Each of these e-commerce sites aggregate online product advertisements. Combined, they represent the *single largest bottleneck* in our dataset and are involved in 58% of all revenue chains from all ad injectors in our synthetic environment.

These shopping programs source creatives from multiple other intermediaries. The most popular is *channelintelligence.com*—a Google owned property that allows brands to market products to other e-commerce aggregators—followed by *pronto.com*. We note that when ad traffic is provided to Google from DealTime or other programs, no provenance information is shared. This prevents Google (and as we discuss shortly, many intermediaries) from identifying traffic that originates from injectors with rudimentary filters alone.

Another notable intermediary in sourcing injected ads, though not sharing any immediate relationships with ad injectors, is *adnxs.com*. This network is run by AppNexus and is involved in 15% of revenue chains. Similarly, we see 5% of all revenue chains go through DoubleClick, a Google ad exchange. We find that 11% of the traffic going through DoubleClick originates via AppNexus, the single largest contributor. These relationships highlight the tangled relationships that underpin injection monetization. Once deceptively sourced traffic enters the ad ecosystem, it becomes intractable to distinguish from legitimate traffic.

Advertisers Negatively Impacted by Ad Injector Traffic: We find a handful of advertisers that are the primary landing page for traffic from ad injectors, a breakdown of which we provide in Table X. We assert no culpability on behalf of these

Injected Ad Domain	Intermediary	% of Ads	Hop After Injector	Hop Before Advertiser
superfish.com	dealtime.com	40%	26%	0%
	pricegrabber.com	34%	22%	9%
	channelintelligence.com	27%	0%	27%
	bizrate.com	23%	23%	1%
	searchmarketing.com	11%	0%	9%
	Other	18%	30%	54%
dealply.com	bizrate.com	67%	57%	2%
	superfish.com	43%	0%	9%
	channelintelligence.com	21%	0%	18%
	amung.com	19%	4%	1%
	clk-analytics.com	17%	0%	0%
	Other	78%	39%	70%
datafast.com	bizrate.com	43%	41%	1%
	frontdb.com	28%	0%	10%
	pronto.com	21%	15%	0%
	channelintelligence.com	19%	0%	17%
	adnxs.com	16%	0%	6%
	Other	67%	44%	66%

Table IX: Top 5 intermediaries involved in sourcing injected traffic, broken down per injected ad domain.

Injected Ad Domain	Advertiser	% of Ads	Avg Hops	Distinct Intermediaries
superfish.com	sears.com	18%	3.3	3
	walmart.com	11%	2.9	6
	kobobooks.com	6%	2.9	3
	target.com	4%	5.1	13
	wayfair.com	2%	4.3	11
	Other	59%	3.8	4,613
dealply.com	target.com	12%	10.5	13
	wayfair.com	5%	9.4	18
	walmart.com	5%	10.4	15
	overstock.com	4%	10.9	11
	sears.com	3%	10.5	18
	Other	71%	8.7	7,146
datafast.com	ebay.com	10%	2.2	60
	target.com	10%	5.8	30
	bizrate.com	5%	1.1	6
	wayfair.com	4%	4.2	15
	sears.com	4%	5.4	15
	Other	68%	5.4	10,692

Table X: Top 5 advertisers terminating our revenue chains, broken down per injected ad domain.

advertisers. Sears and Walmart, whom we observe impacted by all three top injected ad domains, receive 8–29% of our synthetic traffic from each injector. Other popular brands that are impacted include Ebay, Target, Wayfair, and Overstock—all online retailers that specialize in selling consumer products. Although the remaining set of over 3,000 advertisers is heavy-tailed, the top 20 advertisers represent a significant bottleneck, receiving 50% of advertising clicks. Our results highlight that major brands are negatively impacted by ad injection, of which our results only reveal a subset. In particular, our current coverage is biased towards businesses whose product catalogs heavily overlap with Amazon, Walmart, and Google due to the popular product queries we evaluate.

C. Awareness of Advertisers and Intermediaries

We examine the degree of information that advertisers and intermediaries receive about the provenance of ad traffic. To conduct our analysis, we automatically extract encoded "affiliate IDs" embedded in each URL tied to injected ad traffic for the top five advertisers and intermediaries. These affiliate IDs indicate which ad network acquired traffic for which advertiser (facilitating payment and auditing in the ad ecosystem).

We observe that all of the top five advertisers in our dataset have knowledge of their immediate ad relationships (e.g., the previous referrer in revenue chains). Beyond this previous hop visibility, we find only one set of revenue chains that include more granular provenance information. *target.com* and other brands who syndicate advertisements to *channelintelligence.com* are provided sub-syndication affiliate parameters such as "pronto_df" and "pricegrabber_df". These values reveal that traffic previously originated from *pronto.com* and *pricegrabber.com* before arriving at *channelintelligence.com*. Beyond this exception, we never find evidence that advertisers obtain finer-grained provenance information that could help them filter traffic deceptively sourced from ad injection. There is always at least one intermediary between the ad injector and advertiser that omits affiliate IDs tied to injectors.

The story for the top intermediaries in our dataset is different. In particular, we find that intermediaries that share an immediate connection to ad injectors frequently assign a consistent affiliate ID that uniquely indicates an injection library. For some intermediaries this affiliate ID even includes the injection library's domain name. This consistent labeling suggests these early intermediaries have formal business relationships with ad injection entities, or at the very least, awareness of when traffic originates from an ad injector. As a result, programs like DealTime, PriceGrabber, and ShopZilla (detailed previously in Table IX) are best positioned to detect and disincentivize deceptively sourced ads. They serve as the single critical bottleneck before ad injection traffic enters the ad ecosystem and becomes indistinguishable from legitimate consumer interest. Following our analysis, we have begun to reach out to these major intermediaries as well as the brands impacted by ad injection to alert them of the possibility of receiving ad injection traffic.

VII. Case Studies of Novel Ad Injectors

We conclude our analysis of the ad injection ecosystem with several in-depth case studies that highlight the financial incentives and novel technical capabilities of individual injectors.

Superfish: Superfish is a VC-backed startup located in Palo Alto with research and development in Israel. The company reported earnings of $135K in 2010 and $35M in 2013 [16]. The company focuses on visual search offerings with the goal of displaying advertisements for similar products as clients browse the web. Superfish runs its affiliate program via *similarproducts.net*, which provides a single line of JavaScript that affiliates embed in browser traffic for drop-in monetization.

The script has code to support injection into 16,925 websites, presenting additional product advertisements for each supported page. On Google, this yields a bar of images that appears above organic search results. For Amazon, Superfish displays additional products at the bottom of the visible screen area. For other pages, Superfish displays ads as fly-in banners.

In order to fetch advertisements, the Superfish script sends a request to *superfish.com* along with a `merchantName` that indicates the domain the request originates from (e.g., Google); `documentTitle` and `pageUrl` that reports every site a user visits; and `language`, `country`, and `ip` parameters. In response, *superfish.com* returns a list of images, prices, and URLs that the script then injects into the client's DOM. Per our clickchain analysis, we find these ads are predominantly supported by *pgpartners.com*, *bizrate.com*, and *dealtime.com*.

Visadd: Visadd, registered at *visadd.com* under an anonymization service Domains By Proxy, steadily rose in prominence from 0.5% of page views at the start of our measurement to 1.4% at the time of writing. Like other ad injectors, Visadd maintains a blacklist of properties it avoids tampering with, including *google.com*, *facebook.com* and *ads.yahoo.com*.[9] Outside these safezones, the script scans for specific keywords including "add to basket", "free shipping", and "product review" in multiple languages. If found, the script fetches additional payloads to inject advertisements. Beyond Visadd's injection capabilities, we found the script adds event listeners to every link on a page to remotely report user clicks and surfing behavior. When we visited the Visadd website, we came upon an option to uninstall the Visadd injector. However, at the time of writing, all that happens is a call to *function donothing(){};*.

Jsutils, Jscripts, & Webpagescripts: Not all of the ad injection scripts we observe belong to a single identifiable company. In fact, we observe at least four scripts whose sole purpose is to provide drop-in support for several ad injectors simultaneously. We refer to these scripts as *meta-injectors*. Examples in our dataset include *jsutils.net*, *jsutils.com*, *jscripts.org* and *webpagescripts.net*. Each of these scripts deliver support for 17 distinct ad injection affiliate programs, including *jollywallet.com*, *tfxiq.com*, *visadd.com* and *adultadworld.com*; some of the most popular programs we encounter in tampered DOMs.

These meta-injectors maximize the value of injected traffic by supporting both cost-per-click and cost-per-acquisition models. On top of that, the presence of pornographic-oriented injection libraries makes up for other injection libraries that blacklist sexually explicit websites to adhere to commercial ad exchange policies. This profit maximization has many levels, where we find that *tfxiq.com* is configured to fall back onto yet another round of intermediaries, including *adcash.com* and *viglink.com*, if Tfxiq's primary ad relationships fail to provide ad content. The end result is a browsing experience where essentially no matter what action a user takes, ad injectors can profit.

[9]We note that Visadd scripts still appear in client traffic on google.com and other sites; they merely remain dormant and avoid injecting ads.

VIII. Related Work

Fraud and Abuse in the Ad Ecosystem: Prior work has extensively explored the problem of outright fraud and abuse in ad networks. In particular, research has focused on the evolution of botnets towards ad related clickfraud monetization [1], [25] and the impact of botnet interventions on abusive ad traffic [6], [26]. Other forms of abuse include the failures of current ad exchanges to detect distributed clickfraud [33], the ability for compromised routers and opportunistic ISPs to inject ads into users' traffic [29], [34], and the market for impression fraud via ads hidden underneath other content or served in invisible windows [32]. In response, researchers have proposed an array of solutions that attempt to detect fraudulent ad traffic via bluff ads or anomaly detection [9], [10], [15], [17]. We believe future research should pay similar attention to the ad injection affiliate ecosystem in the event malware authors become affiliates (of which we find initial evidence in our work) and the possibility of ad injectors serving pop-under or hidden ads that defraud exchanges.

A sister problem to ad fraud is the emergence of malicious advertisements that leverage ad traffic to fuel malware installs [14]. Of the Alexa Top 90,000, Li et al. found 1% of pages served malicious advertisements [24]. While Dong et al. proposed browser confinement schemes that allow publishers to load ads in restricted sandboxes [11], it falls on publishers to adopt such technologies, which are not yet readily available. Furthermore, while publishers can selectively source creatives from trustworthy ad exchanges to protect their brand, the presence of ad injectors in a user's browsing session place all such discretion solely into the hands of the injector.

Detecting Website Content Modifications: Our approach for detecting ad injection draws on a history of proposed remote verification and enforcement techniques. In the closest work to our own, Reis et al. propose a technique for client-side verification of a webpage's integrity called "web tripwires" [29]. The core idea is for publishers to include an additional piece of JavaScript in their webpages that checks if the client-rendered DOM content matches what publishers expect on their website. In both our and Reis's solution, it remains up to developers to contend with highly dynamic content. Similarly, both schemes suffer from a lack of a trusted communication channel; as a result, adversaries with sufficient privileges can strip out integrity verification code and spoof a valid response. Seshadri et al. proposed a technique for untampered code execution on legacy systems [31], but whether such a technique can be extended to browser kernels to protect execution in the presence of malicious extensions (as well as malicious binaries) remains an open challenge.

IX. Conclusion

In this paper we presented a detailed investigation into the negative impact of ad injection and the ecosystem that supports it. We found that ad injection has entrenched itself as a cross-browser monetization platform impacting more than 5% of unique daily IP addresses accessing Google—tens of millions

of users around the globe. Injected ads arrive on a client's machine through multiple unwanted and malicious vectors, with our measurements identifying 50,870 Chrome extensions and 34,407 Windows binaries, 38% and 17% of which are explicitly malicious. As part of our analysis, we alerted the Chrome Web Store of 192 deceptive ad injection extensions with 14 million user; the Chrome Web Store has since disabled the extensions. Finally, we determined that ad injectors ultimately derive a profit by delivering deceptively sourced traffic to over 3,000 brands. This traffic enters the ad ecosystem through a small bottleneck of e-commerce networks. We have since reached out and alerted the advertisers and intermediaries impacted by ad injectors.

In closing, we argue there is no simple solution for combating deceptive ad injection. Intermediaries, website owners, and browser developers all share an important role. In particular, the handful of e-commerce sites who share relationships with ad injectors are the best positioned to prohibit deceptively sourced traffic and disincentivize the ad injection ecosystem as a whole. For example, Google's ad exchanges expressly prohibit sourcing traffic from ads inserted into websites without the site owner's consent [12]. For website owners, developers can measure their own ad injection levels by executing our client-side measurement, or go one step further and prevent or revert DOM modifications produced by ad injectors. Equally important, if websites switched to HSTS it would prevent network providers and HTTP-only binary proxies from intercepting and tampering with client traffic. Finally, browser developers must harden their environments against side-loading extensions or modifying the browser environment without user consent. Combined, these strategies represent a breadth of technical and financial countermeasures to combat deceptive ad injection.

X. Acknowledgements

We thank Petr Marchenko and Aaseesh Marina for insightful feedback and support in developing our ad injection analysis pipeline. This work was supported in part by the National Science Foundation under grants 1213157, 1237265, and 1237076; by the Office of Naval Research MURI grant N000140911081 and N000141210165; by the U.S. Army Research Office MURI grant W911NF0910553; and by a gift from Google. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Sumayah A Alrwais, Alexandre Gerber, Christopher W Dunn, Oliver Spatscheck, Minaxi Gupta, and Eric Osterweil. Dissecting Ghost Clicks: Ad Fraud via Misdirected Human Clicks. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 21–30. ACM, 2012.

[2] Amazon. Best Sellers Amazon Best Sellers. http://www.amazon.com/Best-Sellers/zgbs, 2014.

[3] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting Browsers from Extension Vulnerabilities. In *NDSS*. Citeseer, 2010.

[4] Nico Black. Superfish Affiliate Summit East. https://www.linkedin.com/groups/Superfish-Affiliate-Summit-East-4376214.S.263442122, 2014.

[5] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-Engineering. In *Proceedings of the ACM Conference on Computer and Communications Security*, Chicago, IL, November 2009.

[6] Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights From the Inside: A View of Botnet Management from Infiltration. In *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2010.

[7] Devin Coldewey. Marriott Puts An End To Shady Ad Injection Service. http://techcrunch.com/2012/04/09/marriott-puts-an-end-to-shady-ad-injection-service/, 2014.

[8] Neil Daswani and Michael Stoppelman. The Anatomy of Clickbot. A. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007.

[9] Vacha Dave, Saikat Guha, and Yin Zhang. Measuring and fingerprinting click-spam in ad networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 175–186. ACM, 2012.

[10] Vacha Dave, Saikat Guha, and Yin Zhang. ViceROI: Catching Click-spam in Search Ad Networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 765–776. ACM, 2013.

[11] Xinshu Dong, Minh Tran, Zhenkai Liang, and Xuxian Jiang. AdSentry: Comprehensive and Flexible Confinement of JavaScript-based Advertisements. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 297–306. ACM, 2011.

[12] DoubleClick. Google DoubleClick Ad Exchange (AdX) Seller Program Guidelines. http://www.google.com/doubleclick/adxseller/guidelines.html, 2014.

[13] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza Basrai, and Peter M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI)*, pages 211–224, December 2002.

[14] Sean Ford, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Analyzing and detecting malicious flash advertisements. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 363–372. IEEE, 2009.

[15] Hamed Haddadi. Fighting online click-fraud using bluff ads. *ACM SIGCOMM Computer Communication Review*, 40(2):21–25, 2010.

[16] Inc. Meet the 2014 Inc. 5000: America's Fastest-Growing Private Companies. http://www.inc.com/profile/superfish, 2014.

[17] Ari Juels, Sid Stamm, and Markus Jakobsson. Combating click fraud via premium clicks. In *USENIX Security*, 2007.

[18] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *Proceedings of the 23rd Usenix Security Symposium*, 2014.

[19] Erik Kay. Keeping chrome extensions simple. http://blog.chromium.org/2013/12/keeping-chrome-extensions-simple.html, 2013.

[20] Erik Kay. Protecting Chrome users from malicious extensions. http://chrome.blogspot.com/2014/05/protecting-chrome-users-from-malicious.html, 2014.

[21] David Kravets. Ad-injecting trojan targets Mac users on Safari, Firefox, and Chrome. http://arstechnica.com/apple/2013/03/ad-injecting-trojan-targets-mac-users-on-safari-firefox-and-chrome/, 2013.

[22] David Kravets. Comcast Wi-Fi serving self-promotional ads via JavaScript injection. http://arstechnica.com/tech-policy/2014/09/why-comcasts-javascript-ad-injections-threaten-security-net-neutrality/, 2014.

[23] Lenovo. Superfish Vulnerability. http://support.lenovo.com/us/en/product_security/superfish, February 2015.

[24] Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng 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.

[25] Brad Miller, Paul Pearce, Chris Grier, Christian Kreibich, and Vern Paxson. What's clicking what? techniques and innovations of today's clickbots. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 164–183. Springer, 2011.

[26] Paul Pearce, Vacha Dave, Chris Grier, Kirill Levchenko, Saikat Guha, Damon McCoy, Vern Paxson, Stefan Savage, and Geoffrey M Voelker. Characterizing Large-Scale Click Fraud in ZeroAccess. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014.

[27] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to us. In *Proceedings of the 17th Usenix Security Symposium*, 2008.

[28] Moheeb Abu Rajab, Lucas Ballard, Noé Lutz, Panayiotis Mavrommatis, and Niels Provos. CAMP: Content-Agnostic Malware Protection. In *Symposium on Network and Distributed System Security (NDSS)*, 2013.

[29] Charles Reis, Steven D Gribble, Tadayoshi Kohno, and Nicholas C Weaver. Detecting In-Flight Page Changes with Web Tripwires. In *Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[30] Sambreel Holdings, LLC vs. Facebook, Inc. http://www.kotchen.com/Sambreel-v-Facebook/PI_Declaration_Miller.pdf, 2012.

[31] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

[32] Kevin Springborn and Paul Barford. Impression fraud in on-line advertising via pay-per-view networks. In *22nd USENIX Security Symposium*, 2013.

[33] Brett Stone-Gross, Ryan Stevens, Apostolis Zarras, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. Understanding Fraudulent Activities in Online Ad Exchanges. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 279–294. ACM, 2011.

[34] Nevena Vratonjic, Julien Freudiger, and Jean-Pierre Hubaux. Integrity of the web content: The case of online advertising. In *CollSec*, 2010.

[35] Walmart. Popular Products - Walmart. http://www.walmart.com/c/popular/, 2014.

[36] WC3. Content Security Policy 1.0. http://www.w3.org/TR/CSP/, 2012.