Understanding Malvertising Through Ad-Injecting Browser Extensions

Xinyu Xing
Georgia Institute of Technology
xxing8@gatech.edu

Wei Meng
Georgia Institute of Technology
wei@gatech.edu

Byoungyoung Lee
Georgia Institute of Technology
blee@gatech.edu

Udi Weinsberg
Facebook Inc.
udi@fb.com

Anmol Sheth
A9.com/Amazon
anmolsheth@gmail.com

Roberto Perdisci
University of Georgia
perdisci@cs.uga.edu

Wenke Lee
Georgia Institute of Technology
wenke@cc.gatech.edu

ABSTRACT

Malvertising is a malicious activity that leverages advertising to distribute various forms of malware. Because advertising is the key revenue generator for numerous Internet companies, large ad networks, such as Google, Yahoo and Microsoft, invest a lot of effort to mitigate malicious ads from their ad networks. This drives adversaries to look for alternative methods to deploy malvertising.

In this paper, we show that browser extensions that use ads as their monetization strategy often facilitate the deployment of malvertising. Moreover, while some extensions simply serve ads from ad networks that support malvertising, other extensions maliciously alter the content of visited webpages to force users into installing malware. To measure the extent of these behaviors we developed Expector, a system that automatically inspects and identifies browser extensions that inject ads, and then classifies these ads as malicious or benign based on their landing pages. Using Expector, we automatically inspected over 18,000 Chrome browser extensions. We found 292 extensions that inject ads, and detected 56 extensions that participate in malvertising using 16 different ad networks and with a total user base of 602,417.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services; K.6.2 [**Installation Management**]: Performance and usage measurement; K.6.5 [**Security and Protection (D.4.6, K.4.2)**]: Invasive software (e.g., viruses, worms, Trojan horses)

General Terms

Security, Design, Experimentation

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741630.

Keywords

Malvertising; Browser Extension; Adware

1. INTRODUCTION

Online advertising is a powerful way to deliver brand messages to potential customers. To monetize their online services and applications, most modern websites act as ad *publishers* and reserve ad space on their web pages where online ads are displayed to their visitors. Ad networks work as brokers between advertisers and publishers. Joining an ad network frees websites from having to set up their own ad servers and invest in tracking software. Consequently, some ad networks attract a very large number of publishers and produce huge revenues.

As online advertising became increasingly popular and pervasive, miscreants have started to abuse this convenient channel to conduct malicious activities. Malvertising is one of such activities, where an attacker maliciously uses advertising to distribute various forms of malware. Malvertising can result in serious consequences, because an attacker can purchase ad space to publish malicious ads on many popular websites. Therefore, the maliciously crafted content could reach a very large audience. In addition, users may be unaware of the fact that they could encounter malicious content while browsing highly reputable websites, which may put them at an even higher risk.

Recent reports indicate that some ad networks have started to offer browser extension developers an opportunity to monetize their "free" work [3, 9]. In other words, some ad networks are extending their business by connecting advertisers to browser extension developers. Given this new business trend and little studies on malvertising in this new context, we conduct a comprehensive study on malvertising activities of ad-injecting extensions, which insert ads on webpages without user consents. In particular, we aim to answer a number of fundamental questions: How many ad-injecting extensions are present on popular repositories like the Chrome Extension store? How many extensions are conducting malvertising activities? What are the characteristics of malvertising via ad-injecting extensions?

To answer the aforementioned questions, we developed `Expector` (Extension Inspector), a tool that is able to automatically identify ad-injecting extensions. We ran `Expector` on extensions

served by the Chrome Extension store, and observed several interesting behaviors. First, compared to ads "naturally" published (without any browser extension intercession) by popular websites, ad-injecting extensions tend to serve a larger fraction of malicious ads. We attribute this to the observation that popular websites partner with large reputable ad networks, whereas extensions utilize smaller ad networks that devote insufficient efforts to identifying and mitigating malicious advertisers. Second, in contrast to previous studies that found only a low fraction of malicious ads, we observed that some ad networks serve only malicious ads. This is presumably because these ad networks and miscreants collude to conduct malvertising, or because miscreants effectively own these ad networks and use them for serving malicious ads. Finally, we found that ad-injecting extensions can make malvertising more detrimental, because some ad networks are unscrupulous on abusing the privileges that ad-injecting extensions offer.

In summary, the main contributions of this paper are as follows:

- We present a comprehensive study that exposes and quantifies the extent to which browser extensions facilitate malvertising. Furthermore, we show that the high privileges granted to extensions are often abused to increase the likelihood of infecting users with malware, e.g., through click hijacking.
- We present `Expector`, a measurement framework that automatically identifies browser extensions that inject ads on webpages. Our evaluation shows that `Expector` has both low false positives (3.6%) and low false negatives (3%).
- We ran `Expector` on all extensions available on the Chrome extension store (almost 18,000), and identified 292 extensions that embed ad-injection capabilities. Of these, 56 deliver ads that lead users to sites that host malware, and have an overall user base of 602,417. Furthermore, we found 16 unique ad-network domains that inject malicious ads, leading users to 117 distinct malware-serving domains.
- We show that a user installing an ad-injecting extension is more likely to be exposed to malvertising threat, compared to users that do not install such browser extensions.

The rest of this paper is structured as follows. We begin with the discussion of related work in Section 2. In Section 3 we provide essential background. We detail the design of `Expector` and show the results from applying it to the Google Chrome extension store in Section 4. We then perform a detailed study for malvertising extensions identified by `Expector` in Section 5 and 6. Finally, we conclude the paper in Section 7.

2. RELATED WORK

In this section, we discuss three lines of work most related to ours – (1) malvertising, (2) misbehaving browser extension, and (3) adware.

Malvertising. Malvertising has been a growing concern for many ad networks. Prior research on this threat has shown the rampancy of malvertising through normal Web browsing [21, 26, 28, 30]. Provos *et al.* [28] studied various channels for distributing drive-by downloads. The authors found that 2% of the landing sites deliver malware via advertisements, which were often reached through multiple ad syndication. Ford *et al.* [21] studied malware distribution through Flash ads and developed a tool for malicious Flash ad detection. According to recent studies [26,30], researchers found that about 1% ads lead users to malicious content. In addition, Zarras *et al.*observed that smaller ad networks are more prone to serving malicious advertisements, and ad arbitration (syndication) process can facilitate the distribution of malicious advertisements. Different from these previous studies, we focus on unraveling those characteristics of malvertising unique to ad-injecting browser extensions.

Browser Extension. Though not considered malicious, ad-injecting activities of browser extensions are misbehaving practice. A series of issues in misbehaving browser extensions have been studied for years, mostly focusing on preventing user data leakage through malicious extensions [20,24,25,27,29] and protecting against privilege abuse of extensions [22]. The security model of Chrome extensions was criticized in [27]. The authors showed that extensions introduce attacks on the Chrome browser itself, and proposed a enforcing micro-privileges at the DOM element level. Egele *et al.* [20,24] proposed several detection solutions to spyware that silently steals user information. By leveraging both static and dynamic analysis techniques the authors show that it is possible to track the flow of sensitive information as it is processed by the web browser and any extension installed. [22] showed that many Chrome extensions are over-privileged, which can potentially cause security risks. To address the shortcomings of existing extension mechanisms, the authors propose a comprehensive new model for extension security. Our work differs from these studies because we focus on a little studied yet important issue in misbehaving browser extensions – ad injection.

A pioneering work in this problem space is Hulk [23], a dynamic analysis system that automatically detects Chrome browser extensions with malicious behaviors. Using Hulk, Kapravelos *et al.*found thousands of suspicious extensions including those with ad-injecting practice. Despite effectiveness on identifying misbehaving browser extensions, Hulk cannot provide us only with a corpus of ad-injecting extensions for studying their malvertising activities because it is designed for identifying various misbehaviors in Chrome extensions. In addition, our study needs not only a tool to identify ad-injecting extensions but also an automatic solution to categorize ads placed by these extensions. Considering the limitation of Hulk as well as our unique requirement, we built a customized measurement tool – `Expector`– to facilitate our study.

In addition to Hulk, other tools have been recently developed to assist users in identifying ad-injecting Chrome extensions [6, 7]. These tools operate by comparing extensions installed by a user against a list, oftentimes crowd-sourced, of known adware, and alerting the user if such extensions are found. Such an approach is limited by the accuracy and completeness of its list, and cannot handle new adware that was not yet reported. Therefore, our study does not rely on manually curated lists.

Adware. As an ad-injecting browser extension is one instance of adware, analysing malvertising activities of ad-injecting extensions also falls under the category of the study on adware. Edelman *et al.* [18] provide an overview of the adware ecosystem, studying the ad networks, exchanges, and practices used by ad injectors. In a more recent work [19], the same authors study fraud in online affiliate marketing, and how adware takes part of such frauds. Both works focus more on the business aspects of the ecosystem, and unlike our work, they do not attempt to conduct a comprehensive analysis of adware, but instead extract the participants of manually selected adware extensions.

3. AD-INJECTING EXTENSIONS

Browser extensions are programs that enhance the functionalities of the browser. These programs are written using a combination of HTML, JavaScript, and CSS, and are typically hosted in online stores, such as the Chrome web store [10] and Mozilla Ad-

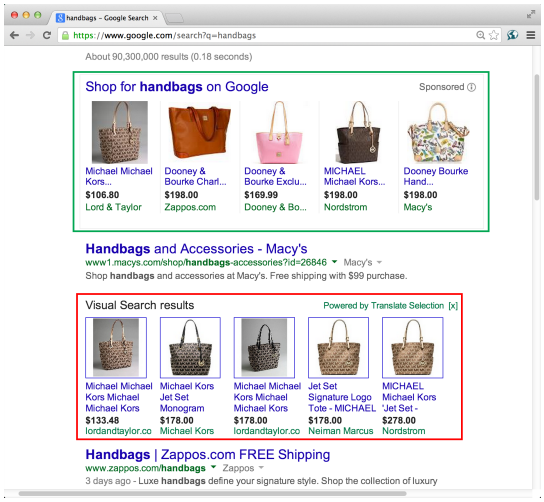


Figure 1: A Google search injected with ads from "Translate Selection" Chrome extension.

Ons store [8]. Extensions interact with the current webpage loaded in the browser by injecting JavaScript to read or modify the DOM structure of the webpage, communicating with external servers using `XMLHttpRequest`, and leveraging the browser APIs and features. The permissions requested by the extension are listed in a manifest file that is reviewed by the user at installation time. As is evident from this description, browser extensions hold significant privileges, thus leaving the door open to malpractices and security and privacy risks. In the rest of the section we describe three primary types of ad injection malpractices that we observed through our study of Chrome browser extensions. Then, we briefly introduce JavaScript libraries used by ad-injecting extensions.

3.1 Ad Injection Practices

Ad Injection on Search Result Pages. Consider the screenshot in Figure 1, showing the result page of a Google search for the query "bags". The user has installed the "Translate Selection" Chrome extension whose primary purpose is to help users quickly translate the selected text between different languages. However, this extension bundles with it the functionality to inject ads in the search results returned to the user. The top highlighted region in Figure 1 shows the standard Google sponsored ads. The bottom highlighted region shows another set of ads of bags with links to online stores.

Ad Injection on Retail Websites. Another type of ad injection practice is related to injecting ads on webpages related to online retail (amazon.com, ebay.com, etc.). When a potential shopper browses a product on an online retail website, the extension sends the context of a browsing session to a third-party advertisement service, which retrieves similar or related products. These products are shown in the form of an ad to the user, overlayed on the existing website content.

Ad Injection on Unrelated Websites. The third type of ad injection practice consists of extensions that aggressively insert ads on almost every webpage that the user browses. This practice can often degrade the user's experience by shown pop-ups or other forms of annoying ads.

3.2 JavaScript Libraries for Ad Injection

It is relatively straightforward for extension developers to monetize their extensions through ad injection. Similar to existing ad networks and ad exchanges, there exists a thriving market of ad networks that provide JavaScript ad injection libraries for extension developers to integrate with their applications. These libraries inject ads on webpages by modifying the DOM structure of the HTML and inserting additional HTML `iframe`'s that contain the injected ad content. The ad provider may trigger the ad injection only on specific websites (e.g., retail websites), and/or inject ads when a user performs a specific operations (e.g., mouse hover on a product image).

4. EXPECTOR

In this section, we provide details for the design and implementation of `Expector`, our browser extension analysis and measurement framework that aims to automatically detect and characterize ad-injection practices used by browser extensions. We then discuss how we use `Expector` to detect ad-injecting extensions (with either "regular" ads or malvertising) in the Chrome Extension store.

4.1 Design

Identifying Triggering Websites. Some ad-injecting extensions inject ads only when the user visits a specific set of websites. Therefore, `Expector` needs to identify the websites that may trigger the ad injection functionalities of each extension to be analyzed. To identify such websites, `Expector` performs static analysis of the extension's code. As some of the ad injection code may be loaded from third-party library at runtime, `Expector` also spawns a browser with the extension installed to intercept all the scripts transmitted via the network. More specifically, `Expector` installs a virtual proxy between the browser's JavaScript engine and the extension. Using the remote debugging protocols supported by browsers (e.g., Chrome [12]), the virtual proxy intercepts all the function invocations related to JavaScript executions. Once the JavaScript code executed by the extension is obtained, it searches for references to one of the ten most popular top-level domains (TLDs) [17] (e.g., `.com`, `.net`, `.org`). We limit the search to these top-ten TLDs because we assume that extensions will most likely target popular websites, almost all of which are registered under the top-ten TLDs.

Triggering Events for Ad Injection. Besides injecting ads based on websites the user visits, extensions also inject ads on specific user generated events. For example, the extension code may register an event listener on a product image of a retail site and listen to the "MouseOver" event. The callback for injecting ads of similar products is only executed when the user hovers his mouse on this image. In order to effectively identify ad injection extension, `Expector` tries to trigger as many as possible potential events that the extension might be interested in. To this end, for all DOM elements that have a JavaScript event handler to process user events, like onClick, onMouseOver, etc., we instruct the browser to execute the corresponding JavaScript function.

A caveat to the above described event trigger mechanism is that some extensions inject ads only after some time has elapsed since the user installed the extension. We assume these practices are used to reduce user dissatisfaction from the injected ads and to reduce the chance to be detected by an extension reviewer. We therefore "tricked" these extensions by setting the system clock before we install an extension to a random long enough time backwards, then install the extension, launch the browser and then set the system clock back.

Identifying Extension-Injected Elements. As a first step for detecting injected ads, the measurement framework should identify the suspicious DOM elements that are potentially inserted by the

extension for ad injection. A naive approach is to directly compare the HTML source of two different pages – one with the extension loaded, and another one without. However this approach will result in a large number of false positives as most webpages load a large amount of dynamic content that can be served by different hosts on each reload of the webpage. For example, each reload of a webpage can potentially result in ads served from different ad providers.

To address this, Expector uses *both* the DOM structure as well as the content of the DOM elements to identify potential extension-injected elements. This is achieved by the following steps:

1. We assume that all hosts that serve content on the webpage *without* the extension installed are trusted and generate a list of trusted host names by loading the webpage multiple times without the extension. The reason we trust such existing hosts is that extensions use different ad networks than mainstream websites, mostly due to restrictions enforced by the large ad networks [2].
2. We load the webpage using two instances of the browser (with and without the extension loaded) and record the DOM tree.
3. We trigger user events in the instance with extension installed as described above.
4. The DOM tree structure is flattened into a list of elements by performing a pre-order traversal over the corresponding DOM tree, i.e., for each node in the list, its descendants are located at its right side and its ancestors are at the left side of the node. Each item in the list consists of the tag name of the DOM element and the domain associated with the DOM element (if such domain exists). For example, an iframe tag like <iframe src='http://www.amazon.com/product/B00BWYQ9YE/'/> is encoded as a two tuple ('iframe', 'www.amazon.com') after transformation.
5. In order to compute the difference between the two lists, we used a variant of the *longest common subsequence* (LCS) algorithm. The modified LCS algorithm outputs nodes that are present only in the webpage loaded with the extension installed.
6. Finally, these elements are further processed and only the elements whose host name is not presented in the trusted list are labeled as potential injected ad elements. Although extensions might use domains in the trusted list to serve ads, we find in practice it is rare as popular and reputable ad networks (e.g. Google AdSense) disallow their usage in extensions [2].

The above process enables us to identify DOM elements that are only added by the extension. As DOM tree manipulation might be a legitimate functionality for some extensions, Expector needs to apply further checks to identify DOM elements that are ads.

Identifying Extension-Injected Ads. A unique characteristic of online ads is that users who click the ad are usually redirected to multiple other sites before eventually reaching the landing page. These redirections enable ad networks to monitor ad clicking for analytics and billing purposes. This redirection pattern is unique to ads and simply detecting it is sufficient to identify that the corresponding DOM element is advertisement. We use these observations for distinguishing ad elements from other non-ad elements.

To this end, Expector visits the ad landing pages potentially associated with the identified DOM elements through redirects. For all elements that have a URL associated with them, we instruct the browser to visit the URL. Note that we ignore iframe's, since the URL points to the source of the iframe. For the remaining elements (iframe's or elements with no URLs), we instruct the browser to trigger a click event to emulate a user clicking inside the element. For all the above cases, if the element contains an ad, the advertiser's website will be loaded usually through a series of redirects, first to one or more ad networks, and then to the ad landing page.

To detect such redirection patterns, we implemented a lightweight Chrome extension, which is loaded before the other extension is installed, that logs all HTTP requests made by the browser. HTTP requests issued as a result of the above process are analyzed for searching redirections. A redirection is detected when a visit (click) results in more than one domain in the traffic trace, and the last domain (e.g., the ad landing page) is not the same as the original web site. If such redirection is identified, the extension is labeled for further inspection.

4.2 Implementation

We implement Expector using Node.js [11] and Selenium [14]. We use Node.js to spawn a Chrome instance and load a Chrome extension for pre-parsing (identifying triggering websites). Using the remote debugging protocol of Chrome, we configure a virtual proxy working as a Chrome Developer Tool that intercepts all the JavaScript function invocations. Specifically, we listen on all *Debugger.scriptParsed* events and log all the JavaScript code parsed by the V8 JavaScript engine. The rest of the components of Expector are implemented with Selenium and the LCS algorithm is implemented in Python.

To ensure fast and scalable processing of a large number of Chrome extensions, we deploy Expector across 60 Linux Debian 7 virtual machines running on a 32 core server with 128 GB RAM. This setup enabled us to process all 18,030 extensions in the Chrome webstore in less than three days.

4.3 Evaluation

In order to assure the completeness of our study on the Chrome extensions store, we evaluate the accuracy of Expector to detect ad injecting extensions by characterizing the false positive and negative rates using two different datasets:

Measuring false positives. We evaluate the false positive rate of Expector by testing all the available extensions on the Chrome web store followed by manual verification. To this end, we developed a crawler that downloads all extensions that are listed on the Chrome Web Store [10] along with the extension's meta data, i.e., developer account, extension category, number of active users, rating, description, and user reviews. We ran our crawler during March 2014, and obtained 18,030 Chrome extensions[1].

Out of the 18,030 extensions downloaded from the Chrome web store, 108 extensions (0.6%) failed to be processed by Expector. By inspecting the code of these extensions we found that these extensions utilize native binary code [13] that was not ported by the extension developer to Linux. All other 17,922 extensions were successfully evaluated by Expector using the setup within three days highlighting the ability of Expector to scale to a large number of extensions.

Expector reported 303 extensions as adware, which accounts for 1.7% extensions on Google Chrome store. We manually installed each of these and inspected their source code, and found that 292 of these are indeed adware, which is notably higher (9×)

[1]In [23], Kapravelos *et al.*evaluated Hulk using a larger dataset containing 48,332 Chrome extensions presumably because they collected their extension samples through different channels.

than the crowd-sourced approach used by Extension Defender [6]. This indicates a very low false positive rate of 3.6%. We further analyzed the 11 extensions that Expector incorrectly identified and find that Expector flagged them as users were re-directed to webpages controlled by the extension developer that contained ads. For example, parental control extensions like *Anti-Porn Pro* and *No Xvideos* maintain a blacklist of websites. When users navigate to these webpages, the extension re-directs the user to a webpage hosted by the extension developer showing a warning message along with a few ads. This webpage hosts ads which are detected by Expector and consequently the corresponding extension is flagged as adware.

Measuring false negatives. In order to study the false negative rate of Expector we use a crowdsourced list of Chrome extensions that are tagged as adware. This list is provided by Extension Defender [1] that enables users to submit extensions as potential adware which are reviewed and verified manually by the curators of this list. In March 2014, this Extension Defender had 78 Chrome extensions that were tagged as adware.

We first manually verify the crowdsourced list of extensions tagged as adware. Surprisingly, we found that out of the 78 extensions only 34 of them were manually verified as adware. This initial analysis highlights the limitations of the crowdsourced approach where curators cannot manually check every update to the extension source code and verify user complaints. Consequently, we postulate that these 44 extensions may have disabled the ad injection functionality after being flagged as adware, and Extension Defender failed to remove them from their list.

Out of the 34 manually verified adware extensions, Expector was able to correctly detect 32 of these. We further analyzed the two extensions that Expector missed. The first one uses a Windows DLL file, which is not compatible on Linux, but could have been processed if Expector was deployed on Windows. We do not count this as a false negative. The second extension operates on Facebook, and uses a more sophisticated triggering method. It requires the user to visit Facebook.com, scroll to the bottom of the page, and wait for 10 seconds before injecting ads. Expector correctly identified the triggering website and the timeout event, but failed to detect the required scroll event. These complex triggering events are difficult to identify. However, as our analysis shows, they are quite rare as they target a very specific user interaction, thus reducing the number of ads the extension injects (and lowering their revenue). In summary, Expector has a low false negative rate of 3.0% (1/33).

5. STUDYING MALVERTISING IN A CONTROLLED ENVIRONMENT

In this section, we expose and characterize malvertising activities by interacting with those ad-injecting extensions that Expector identifies.

5.1 Methodology

As described in Section 4, Expector detects 292 ad-injecting extensions. To study their malvertising activities, we need to be able to classify the ad as being malicious, i.e., an ad that leads the user to a malware hosting domain. To this end we use the following process.

Each ad-injecting extension identified by Expector, is automatically installed in a browser and instrumented so that it injects ads. Expector then browses a set of pages and activates a "click" event on all ads (both injected by the extension and those appearing on the page regardless of the extension) that appear during this browsing session and logs the landing pages.

All of the "safe browsing" APIs that we tested (including Google Safe Browsing API and www.bluecoat.com) failed to detect the majority of ad landing pages in our study, probably because the ad landing URLs are constructed dynamically. As a result, we visit each of the ad landing pages and observe whether it hosts an executable or contains links that enable to download an executable. In such cases, the executable is downloaded and uploaded to an online service (VirusTotal [16]) to check whether it is a malware. Based on the returned results we classify the ad as either malicious or non-malicious.

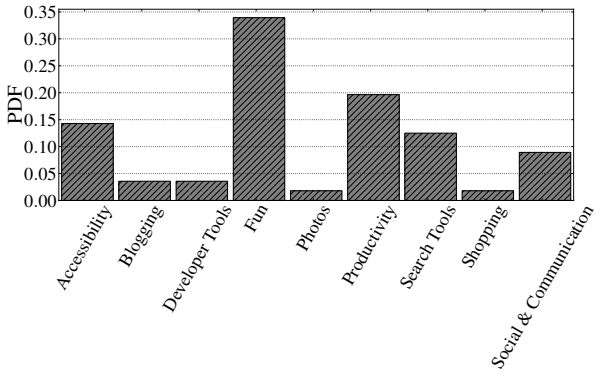


Figure 2: The distribution of ad-injecting Chrome extensions with malvertising practice across different categories.

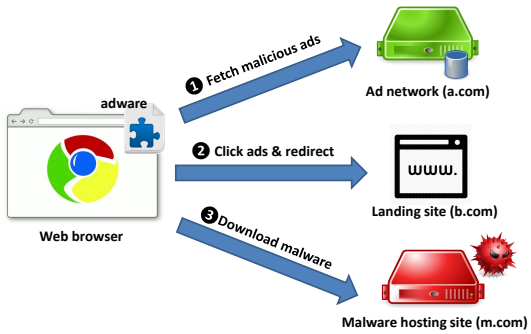


Figure 3: The malvertising flow of an ad-injecting extension, showing a common pattern of three different domains – 1) ad network domain for serving the ad, 2) a landing domain reached after the ad is clicked, and 3) the domain hosting the malware executable.

5.2 Results

We studied malvertising activities by setting the set of pages that Expector visits to the top-1000 Alexa websites. We observed that ad-injecting extensions operate similarly to standard websites and use APIs provided by an ad network for injecting ads. We group together ad-injecting extensions based on the ad networks which they use for fetching ads.

As described above, Expector provides all the ads appearing on each website (separated to organic ads and those injected by the extension), and we classified the ads injected by the ad-injecting extensions based on whether an ad redirects a user to a landing page where malware can be downloaded. As shown in Table 1, we found that 16 out of the 67 (24%) ad network domains deliver malicious

Table 1: Characteristics of ad networks, i.e., domains that serve ads to the ad-injecting extensions in our study.

Ad network domain	Number of extensions	Number of malware hosting domains	Fraction of malvertising	Number of unique ads
toparcadehits.com	1	66	1.0	119
a.kaytri.com	5	50	1.0	137
my-uq.com	8	4	1.0	12
go.webfind.pw	15	1	1.0	3
pchealthcheckup.net	15	3	1.0	3
onlinewebfind.com	14	4	1.0	11
premiumvideoupdates.com	17	2	1.0	12
feeds.webmakerplus.info	8	14	1.0	27
search.buzzdock.com	7	1	1.0	1
simplyfwd.com	5	2	1.0	3
search.privitize.com	2	4	1.0	7
www.freedailydownload.com	1	1	1.0	7
speedtestbeta.com	2	34	1.0	61
adfishmedia.go2cloud.org	1	7	0.640	46
nym1.ib.adnxs.com	17	2	0.153	54
lax1.ib.adnxs.com	2	1	0.490	14

Table 2: The top-5 most popular ad-injecting extension that delivered malicious ads. Notice that the top four hijack all mouse-click events and constantly serve malicious ads.

Ad-injecting extension name	User base	Fraction of malvertising	Malicious activities
HD-Total-Plus	183,470	1.0	Hijacking click events
DownloadTerms	114,851	1.0	Hijacking click events
BetterSurf	97,250	1.0	Hijacking click events
Expresso Smileys and Emoticons	3,392	1.0	Hijacking click events & adding hyperlinks
memeticons2	2,358	0.49	Simple ad injection

ads (i.e., engage in malvertising) – when clicking ads delivered by these ad networks, users are redirected to landing pages that manipulate the users into downloading malicious executables. These 16 ad network domains are used by 56 extensions out of the 292 ad-injecting extensions that we studied (19%). Overall we found 16 unique executables, all of which are labeled as *Trojan*.

Figure 2 plots the distribution of the 56 malvertising extensions across the different extension categories. Interestingly, the majority of these extensions (over 35%) fall under the "Fun" category, perhaps trying to capture many users that might be less tech-savvy and are more likely to be manipulated into installing malware.

For these 16 ad network domains, we note that malicious executables are typically not hosted on the same site as the landing pages of the malicious ads. A common case we observed is depicted in Figure 3 where the ad-injecting extension fetches ads from an ad network (a.com). When a user clicks the ad, she is redirected to a landing page residing on a different domain – a malicious ad landing domain (b.com). Finally, the malware itself is typically hosted on a third domain (m.com). We assume that this is because separating ad landing domains from malware hosting domains makes it more difficult for browser companies, such as Google (via their Safe browsing API), Firefox and Apple from labeling the malicious landing sites as malware site.

Table 1 lists the 16 ad network domains, i.e., the sites that deliver ads to extensions that are used for malvertising. We observe that they are not the popular ad networks, with the exception of `nym1.ib.adnxs.com` and `lax1.ib.adnxs.com` that use AppNexus, a large and very popular ad network [5]. The large ad networks generally prohibit using their network for displaying ads on domains not owned by the publisher, essentially disallowing extensions to inject ads on arbitrary domains [2]. Furthermore, they ban all forms of malvertising [4]. It is thus surprising that AppNexus hosts malvertising. While this might be attributed to AppNexus not being strict enough regarding their advertisers, it might be the result of a compromised legitimate advertiser – clicking the ad redirects to an advertiser's page that might have been compromised by hackers, and they redirected to a malicious page hosting malware [15].

Table 1 also lists the number of domains actually hosting a downloadable malware. As the table shows, some host the malware in many different domains, reaching well over 30. We assume that this is to make it more difficult for browser companies to mark these domains as malicious. Finally, the table also provides the fraction of malicious ads delivered by each domain during `Expector`'s visits to the top-1000 Alexa websites. A key observations is that the extension-inserted ads are mostly malicious for almost all the 16 ad network domains – 13 out of the 16 ad network domains delivered only malicious ads leading users to download malware. This is in clear contrast to the organic ads embedded on webpages, where we did not observe any malicious ad. This is presumably because the top-1000 alexa websites entertain business with large reputable ad networks, which have already deployed effective defense mechanisms against malvertising and serve nearly no malicious ads. In contrast, Chrome extensions employ small – maybe malicious – ad networks, which may collude with malicious advertisers, primarily serve malicious ads and offer extension developers higher rate than those reputable ones.

Table 2 provides details for the top-5 most popular ad-injecting extension we identified as participating in malvertising. The table shows that 4 out of 5 of these extensions engage solely in malvertising, meaning they only serve malicious ads to users. Furthermore their user base is fairly large, with hundreds of thousands of active victim users.

Table 3: The HTTP request patterns originated from ad-injecting extension for each ad network. The table shows the purpose of the request (fetch an ad or report that a user clicked on an ad), the field used to identify the affiliate (extension), and the method used by the ad network to track the user.

Ad network	Request pattern	Request type	Affiliate ID field
txtsrving.info	cdncache1-a.akamaihd.net/loader/...pid=...	fetch ad	pid
	i.txtsrving.info/kwdu?...subid=...	fetch ad	subid
	p.txtsrving.info/click?..subid=...	click	subid
superfish.com	www.superfish.com/ws/findByUrl.action?...userid=...dlsource=...	fetch ad	dlsource
	www.superfish.com/ws/offerURL.action?...userid=...dlsource=...	click	dlsource
imgclck.com	www.imgclck.com/supp0rt/www/delivery/afr.php?...&beacon=...	fetch ad	beacon
xtensionplus.com	xtensionplus.com/display.htm?...&pi=...	fetch ad	pi

Table 4: Relationships between malicious ad network domains and corresponding malware hosting domains.

Ad network domain	Average correlation between ad network domains & identified malware domain lookups	Average correlation between ad network domains & other malware domain lookup	Fraction of overlapping IPs per day
a.kaytri.com	0.870	0.220	0.006
my-uq.com	0.810	0.121	0.009
go.webfind.pw	0.819	0.161	0.470
pchealthcheckup.net	0.840	0.122	0.153
premiumvideoupdates.com	0.925	0.063	0.098
feeds.webmakerplus.info	0.851	0.209	0.153
simplyfwd.com	0.917	0.082	0.326
search.privitize.com	0.848	0.201	0.332
www.freedailydownload.com	0.676	-0.419	0.286

5.3 Summary

During our study, none of the ads originally embedded on the top-1000 Alexa websites were malicious, whereas we found that roughly 4% of extension-injected ads on these websites were malicious. This implies that a user installing an ad-injecting extension is more likely to be exposed to malvertising, even though she only visits highly popular websites.

6. MALVERTISING IN THE WILD

In this section, we continue to explore malvertising activities of ad-injecting extensions. In particular, we are interested in the following questions. (1) For many ad networks shown in Table 1, why are the served ads primarily malicious, whereas a prior study [30] indicates the proportion of malicious ads on an ad network is up to around 40%? (2) We demonstrated that a user could be tricked into clicking malicious ads injected by extensions, visiting the landing pages and eventually downloading malware; however, are there any users actually tricked into downloading malware in the real world? (3) If so, how likely do users click malicious ads injected by extensions and eventually download malware? Using datasets obtained from large real-world networks, we answer these questions in the subsequent sections.

6.1 Dataset

In order to perform the study on malvertising through extensions, we use two datasets provided to us by a large ISP network, a security company, and two universities, all of which monitor their users' network traffic.

HTTP Traffic. We obtained the HTTP traffic log of tens of thousands of users from two university networks and tens of enterprise network during the entire month of May 2014. For each HTTP request, we obtained a source identifier, the destination IP address, and the complete HTTP request header including the user's cookie identifier (if present), requested URL path and referrer.

DNS Traffic. A large ISP provided us the DNS queries performed to the domains we study during the 30 days in May 2014. The DNS traffic log contains A-type DNS records from around 20 million machines located in the US. Each DNS query contains an identifier for the requesting host and the requested domain.

6.2 Analysis of Ad Networks

To tackle the first question described above, we restrict our study to the ad networks shown in Table 1. Then we identify patterns in the URLs that each ad-injecting extension generates by inspecting the source code, and extract the affiliate IDs which are unique IDs used for tracking ad-injecting extensions. Table 3 lists some traffic pattern examples used by ad-injecting extensions, along with the fields we use to identify the affiliate IDs (leading us to the corresponding ad-injecting extension) contained in the HTTP requests.

We used the aforementioned HTTP dataset and observed the HTTP traffic to each ad network. For the ad networks shown in Table 4, we surprisingly found that, all HTTP traffic to these ad network matches the traffic patterns that the corresponding extensions generate. Furthermore, we inspected the referrer fields of the HTTP traffic by visiting the referrer pages. We searched the corresponding ad network domain names in the referer pages and found that none of the referrers contain these ad network domains. These observations imply that the ad network domains that appear in the traffic logs are accessed due to displaying an ad injected by an extension rather than some other method (e.g., email links or organic web ads). In other words, the ad networks in Table 4 only entertain business with advertisers and extension developers rather than website owners (publishers). Presumably, this is the reason why the prior study does not identify ad networks that primarily serve malicious ads. In addition, this serves as an evidence that some ad

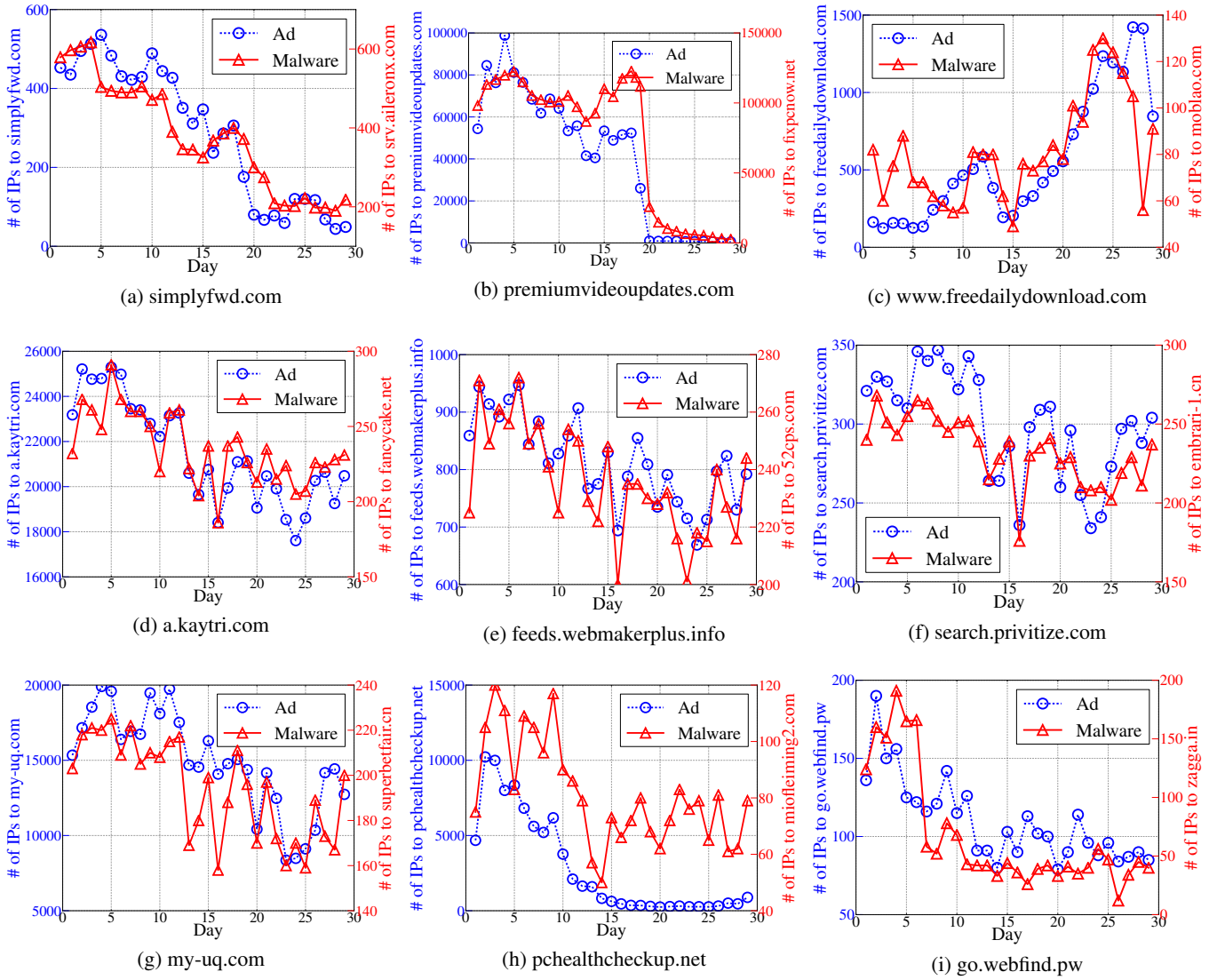


Figure 4: Number of unique sources resolving malicious ad network domains ("Ad") and the number of unique sources resolving one of the corresponding malware hosting domains ("Malware"). The plots clearly show visible positive correlation between the two metrics.

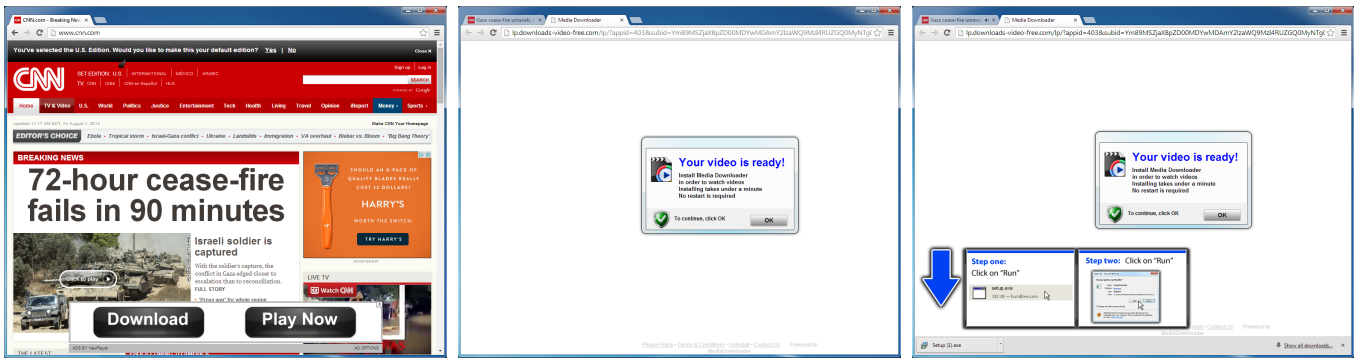
networks are malicious and legit websites never have business with them.

6.3 Correlation Between Ad-Injecting Extensions and Malware

We explore the second question by considering our DNS dataset, and looking for correlations between users resolving the ad network domains and those resolving the domains hosting the downloadable malware (meaning, users that actually download the malware, not only clicked an ad and visited the landing domain). We restrict this study to ad networks that we found to be used solely by extensions and participate in malvertising. Table 4 lists the 9 ad networks matching these criteria.

Ideally, we would like to show causality relationship between downloading malware and receiving ads from the extensions. However, to this end one must obtain complete temporal HTTP flows of many users, which we do not have. Instead, we use our DNS dataset and show strong temporal correlation between the users that resolve the ad network domains and the malware hosting domains.

Figure 4 shows the number of unique sources that resolve an ad network domain and one of malware hosting domains a corresponding ad redirects to. Overall, the plots show a clear correlation between the two. Specifically consider the plot corresponding to `premiumvideoupdates.com`, the curves exhibit a similar temporal pattern, but moreover both curves exhibit a sharp decrease in the number sources on the 20th day of the month. This date is actually the date that Google took down a few popular extensions that used `premiumvideoupdates.com` as their ad network domain. While the domain `fixpcnow.net` still existed after the removal of the extensions, the fact that the extension became unusable reduced its visit count to almost zero. Although we cannot claim causality, this example does serve as a strong evidence for causality. Moreover, the inverse causality argument cannot be made, i.e., even if Google marked the domain `fixpcnow.net` as unsafe, bringing its visitors count to zero, this will not have an impact on the ads served by the malicious extension (and thus the number of users resolving its ad serving domain).



(a) Malvertising on cnn.com. (b) Media player updates. (c) Trojan drops.

Figure 5: Extension Plus-HD 1.3 inserts a malicious ad on cnn.com, pop ups media update warning and drops a Trojan on the unwary user machine.

In order to study the correlation beyond a few illustrative examples, we further computed the Pearson correlation values between monthly DNS lookups of each ad network domain and all of its corresponding malware hosting domains. Table 4 shows the average Pearson correlation values for each ad network domain over all its malware hosting domains. The table shows that the average correlation is very high (1 indicates perfect positive correlation and -1 indicates perfect negative correlation).

To strengthen our belief that these correlations can indicate causality, we computed the correlation of DNS resolutions for our ad networks with 44 other malware domains, which are identified as hosting malware by Google safe browsing API, and are not a part of the malware hosting domains that we identified. Table 4 shows that average Pearson correlation values between the ad network domain lookups and these arbitrary malware domain lookup is significantly lower than the correlation with the malware domains we observe through extensions malvertising. This serves as another evidence that the extensions are indeed correlated with the specific malware domains reached through ads injected by the extensions and not to any arbitrary malware hosting domains.

6.4 Malicious Ad Conversion

Finally, we seek to assess the probability that a user of a malvertising extension will download malware. To this end we consider the overlapping source identifiers in our DNS dataset that resolve both an ad network domain and a corresponding malware hosting domain. Recall that the DHCP churn rate of the ISP is extremely low in a daily basis and the majority of the ISP customers that provided this data are home users, it is very likely that the sources that resolve both originate from a single user (host). Overall, the fraction of overlapping source identifiers indicates the infection rate of malware delivered by extensions that use the identified ad network.

Table 4 shows this fraction, ranging from as low as 0.6% to an astonishing 47% (meaning, almost 1 in 2 sources that resolve the ad network domain also resolve a malware domain). In order to better understand the high variance in this fraction we installed the extensions corresponding to each overlap ratio and studied their methods for injecting malicious ads. As expected, we found that the extensions exhibiting low overlap ratios use much more subtle methods that resemble standard online ads, whereas those exhibiting the high overlap ratios use dubious techniques, such as abusing extension privilege to hijack all mouse click events (essentially resulting in a click-through-rate of 100%). The reason that the overlap never reaches 100% is that we measure DNS resolution of the download page, and users oftentimes understand that they are being manipulated and avoid downloading the malware.

For example, Figure 5 shows the consequences of installing a Chrome extension called *Plus-HD 1.3* (now removed from the Chrome extensions store). When the user visits a page, such as `www.cnn.com`, a popup is injected at the bottom of the browser window (not the bottom of the page!). When an unwary user that clicks anywhere on the page (not only on the buttons), a new tab opens up, showing as if a video is ready for watching, but indicates that the user needs to download and install a "Media Downloader". If the user indeed clicks, the malware is downloaded and the user is prompted to execute it. Table 2 provides the details of ad injection behavior for the top popular malicious extensions.

Overall, these overlap ratios indicate that these clearly malicious practices of extension developers actually pay off in the short run, since users that install them cannot really avoid being directed to the download page, and as we show, many of them actually download the malware (hopefully, not all of them install it). In the long run such behavior might not be the best course of action for the attackers because they can easily be detected by many users, reported to the extension store and get quickly disabled and removed from the store.

7. CONCLUSIONS

In this paper, we performed the first in-depth study on malvertising in the context of ad-injecting browser extensions. We showed that the increasing efforts of large ad networks to mitigate malvertising from their networks cause browser extensions to become a new avenue for malicious malvertising activities. We found that users of ad-injecting extensions are more likely to be exposed to the malvertising threat even though they only visit malicious-ad free popular websites. We also found that an ad-injecting extension can significantly facilitate malvertising and make this activity more harmful especially when miscreants abuse the privilege that ad-injecting extensions offer. Though we only observed roughly 20% of ad-injecting extensions that conduct malvertising, other ad-injecting extensions can easily go beyond the grey area because they leave the door open to malpractices. As part of future work we plan on extending `Expector` beyond Chrome, so that it can process extensions of other popular browsers. Furthermore, our study lies on the foundations to the analysis of web apps, which are very similar to extensions for increasingly popular browser-based OSes, such as Chrome OS and FireFox OS.

Acknowledgments

The authors would like to thank the anonymous reviewers and the IBM researcher, Yunhui Zheng, for their help and feedback. This material is based in part upon work supported by the National Science Foundation under Grants No. CNS-1017265, CNS-0831300 and CNS-1149051, by the Office of Naval Research under Grant No. N000140911042, by the Department of Homeland Security under contract No. N66001-12-C-0133, and by the United State Air Force under Contract No. FA8650-10-C-7025. 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 National Science Foundation, the Office of Naval Research, the Department of Homeland Security, or the United States Air Force.

8. REFERENCES

[1] Abusive extension submission. http://extensiondefender.com/submit.php.

[2] Ad placement policies. https://support.google.com/adsense/answer/1346295#Ads_on_the_same_page_or_site_as_another_publisher.

[3] Adware companies are buying up popular chrome add-ons. http://www.omgchrome.com/malware-buying-google-chrome-extensions/.

[4] Anti-malvertising.com. http://www.anti-malvertising.com/.

[5] Appnexus. http://www.appnexus.com/.

[6] Extension defender. http://extensiondefender.com/.

[7] Extshield notifies you if you're running an adware extension. http://lifehacker.com/chrome-protector-notifies-you-if-youre-running-an-adwa-1505371480.

[8] Firefox add-ons. https://addons.mozilla.org/en-US/firefox/.

[9] Get ready, chrome users - you're about to start seeing ads inside of extensions. http://thenextweb.com/google/2012/07/03/get-ready-chrome-users-youre-about-to-start-seeing-ads-inside-of-extensions/.

[10] Google Chrome Web Store. https://chrome.google.com/webstore/.

[11] Node.js. http://nodejs.org/.

[12] Remote Debuggin Protocol, Google Developers. https://developers.google.com/chrome-developer-tools/docs/debugger-protocol.

[13] Saying goodbye to our old friend npapi. http://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html.

[14] Selenium automates browsers. http://docs.seleniumhq.org/.

[15] Tips for publishers. http://www.anti-malvertising.com/tips-for-publishers.

[16] Virustotal. https://www.virustotal.com/.

[17] Web Technology Surveys. http://w3techs.com/technologies/overview/top_level_domain/all.

[18] B. Edelman and W. Brandi. The ad networks and advertisers that fund ad injectors. http://www.benedelman.org/injectors/, 2013.

[19] B. Edelman and W. Brandi. Information and incentives in online affiliate marketing. In *HBS Working Paper*, 2013.

[20] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song. Dynamic spyware analysis. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 18:1–18:14, Berkeley, CA, USA, 2007. USENIX Association.

[21] S. Ford, M. Cova, C. Kruegel, and G. Vigna. Analyzing and detecting malicious flash advertisements. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 363–372, Washington, DC, USA, 2009. IEEE Computer Society.

[22] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. Verified security for browser extensions. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 115–130, Washington, DC, USA, 2011. IEEE Computer Society.

[23] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, Aug. 2014. USENIX Association.

[24] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer. Behavior-based spyware detection. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[25] Z. Li, X. Wang, and J. Y. Choi. Spyshield: Preserving privacy from spy add-ons. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*, RAID'07, pages 296–316, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 674–686, New York, NY, USA, 2012. ACM.

[27] L. Liu, X. Zhang, V. Inc, G. Yan, and S. Chen. Chrome extensions: Threat analysis and countermeasures. In *Proceedings of 19th Network and Distributed System Security Symposium*, NDSS '12, 2012.

[28] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[29] M. Ter Louw, J. Lim, and V. Venkatakrishnan. Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 4(3):179–195, 2008.

[30] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of Internet Measurement Conference*, IMC '14, Vancouver, BC, Canada, 2014. ACM.