Identifying Extension-based Ad Injection via Fine-grained Web Content Provenance

Sajjad Arshad, Amin Kharraz, and William Robertson
Northeastern University, Boston, USA
`{arshad,mkharraz,wkr}@ccs.neu.edu`

Abstract. Extensions provide useful additional functionality for web browsers, but are also an increasingly popular vector for attacks. Due to the high degree of privilege extensions can hold, extensions have been abused to inject advertisements into web pages that divert revenue from content publishers and potentially expose users to malware. Users are often unaware of such practices, believing the modifications to the page originate from publishers. Additionally, automated identification of unwanted third-party modifications is fundamentally difficult, as users are the ultimate arbiters of whether content is undesired in the absence of outright malice.

To resolve this dilemma, we present a fine-grained approach to tracking the provenance of web content at the level of individual DOM elements. In conjunction with visual indicators, provenance information can be used to reliably determine the source of content modifications, distinguishing publisher content from content that originates from third parties such as extensions. We describe a prototype implementation of the approach called ORIGINTRACER for Chromium, and evaluate its effectiveness, usability, and performance overhead through a user study and automated experiments. The results demonstrate a statistically significant improvement in the ability of users to identify unwanted third-party content such as injected ads with modest performance overhead.

Keywords: Web security, Ad injection, Browser extension

1 Introduction

Browser extensions enhance browsers with additional useful capabilities that are not necessarily maintained or supported by the browser vendor. Instead, this code is typically written by third parties and can perform a wide range of tasks, from simple changes in the appearance of web pages to sophisticated tasks such as fine-grained filtering of content. To achieve these capabilities, browser extensions possess more privilege than other third-party code that runs in the browser. For instance, extensions can access cross-domain content, and perform network requests that are not subject to the same origin policy. Because these extensive capabilities allow a comparatively greater degree of control over the browser, they provide a unique opportunity to attack users and their data, the underlying system, and even the Internet at large. For this reason, newer browser extension frameworks such as Chromium's have integrated least privilege separation via isolated worlds and a fine-grained permissions system to restrict the capabilities of third-party extensions [7].

However, extension security frameworks are not a panacea. In practice, their effectiveness is degraded by over-privilege and a lack of understanding of the threats posed by highly-privileged extensions on the part of users [18]. Indeed, despite the existence of extension security frameworks, it has recently been shown that extension-based advertisement injection has become a popular and lucrative technique for dishonest parties to monetize user web browsing. These extensions simply inject or replace ads in web pages when users visit a website, thus creating or diverting an existing revenue stream to the third party. Users often are not aware of these incidents and, even if this behavior is noticed, it can be difficult to identify the responsible party.

While ad injection cannot necessarily be categorized as an outright malicious activity on its own, it is highly likely that many users in fact *do not want or expect* browser extensions to inject advertisements or other content into Web pages. Moreover, it can have a significant impact on the security and privacy of both users as well as website publishers. For example, recent studies have shown that ad-injecting extensions not only serve ads from ad networks other than the ones with which the website publishers intended, but they also attempt to trick users into installing malware by inserting rogue elements into the web page [46, 48].

To address this problem, several automatic approaches have been proposed to detect malicious behaviors (e.g., ad injection) in browser extensions [48, 28, 26]. In addition, centralized distribution points such as Chrome Web Store and Mozilla Add-ons are using semi-automated techniques for review of extension behavior to detect misbehaving extensions. However, there is no guarantee that analyzing the extensions for a limited period of time leads to revealing the ad injection behaviors. Finally, a client-side detection methodology has been proposed in [46] that reports any deviation from a legitimate DOM structure as potential ad injections. However, this approach requires a priori knowledge of a legitimate DOM structure as well as cooperation from content publishers.

Although ad injection can therefore potentially pose significant risks, this issue is not as clear-cut as it might first seem. Some users might legitimately want the third-party content injected by the extensions they install, even including injected advertisements. This creates a fundamental dilemma for automated techniques that aim to identify clearly malicious or unwanted content injection, since such techniques cannot intuit user intent and desires in a fully automatic way.

To resolve this dilemma, we present ORIGINTRACER, an in-browser approach to highlight extension-based content modification of web pages. ORIGINTRACER monitors the execution of browser extensions to detect content modifications such as the injection of advertisements. Content modifications are visually highlighted in the context of the web page in order to *i)* notify users of the presence of modified content, and *ii)* inform users of the *source* of the modifications. With this information, users can then make an informed decision as to whether they actually want these content modifications from specific extensions, or whether they would rather uninstall the extensions that violate their expectations.

ORIGINTRACER assists users in detecting content injection by distinguishing injected or modified DOM elements from genuine page elements. This is performed by annotating web page DOM elements with a *provenance label set* that indicates the prin-

cipal(s) responsible for adding or modifying that element, both while the page is loading from the publisher as well as during normal script and extension execution. These annotations serve as trustworthy, fine-grained provenance indicators for web page content. ORIGINTRACER can be easily integrated into any browser in order to inform users of extension-based content modification. Since, ORIGINTRACER identifies all types of content injections, it is able to highlight all injected advertisements regardless of their types (e.g., flash ads, banner ads, and text ads).

We implemented a prototype of ORIGINTRACER as a set of modifications to the Chromium browser, and evaluated its effectiveness by conducting a user study. The user study reveals that ORIGINTRACER produced a significantly greater awareness of third-party content modification, and did not detract from the users' browsing experience. Our results also suggests that ORIGINTRACER can be used as a complementary system to ad blocking systems such as Adblock Plus [2] and Ghostery [4].

To summarize, the main contributions of this paper are:

– We introduce a novel in-browser approach to provenance tracking for web content at the granularity of DOM elements, and present a semantics for provenance propagation due to script and extension execution. The approach leverages a high-fidelity in-browser vantage point that allows it to construct a precise provenance label set for each DOM element introduced into a web page.
– We implement a prototype called ORIGINTRACER that uses content provenance to identify and highlight third-party content injection – e.g., unwanted advertisements – by extensions to notify users of their presence and the originating principal.
– We evaluate the effectiveness, usability, and performance of our prototype, and show that it is able to significantly assist users in identifying ad injection by extensions in the wild without degrading browser performance or the user experience.

2   Background & Motivation

In the following, we introduce background information on browser extensions, present an overview of advertisement injection as a canonical example of questionable content modification, and motivate our approach in this context.

2.1   Browser Extensions

Browser extensions are programs that extend the functionality of a web browser. Today, extensions are typically implemented using a combination of HTML, CSS, and JavaScript written against a browser-specific extension API. These APIs expose the ability to modify the browser user interface in controlled ways, manipulate HTTP headers, and modify web page content through the document object model (DOM) API. An extension ecosystem is provided by almost all major browser vendors; for instance, Google and Mozilla both host centralized repositories of extensions that users can download at the Chrome Web Store and Mozilla Add-ons sites, respectively.
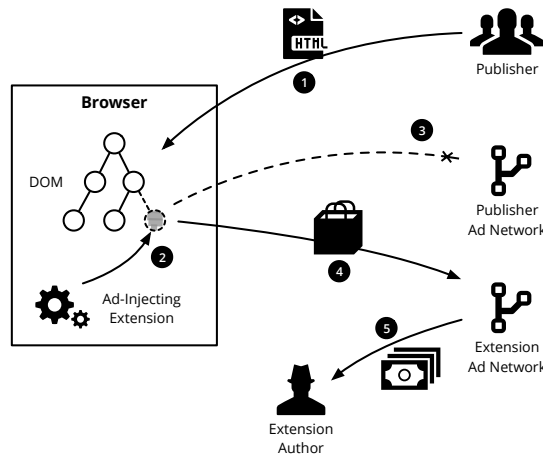
Fig. 1: Overview of advertisement injection. **(1)** The user accesses the publisher's site. **(2)** An ad-injecting browser extension adds DOM elements to display ads to the user, and optionally removes existing ads. **(3)** Ad revenue is diverted from the publisher. **(4)** Ad impressions, clicks, and conversions are instead directed to the extension's ad network. **(5)** Ad revenue flows to the extension author.

2.2 Advertisement Injection

As web advertising grew in popularity, those in a position to modify web content such as ISPs and browser extension authors realized that profit could be realized by injecting or replacing ads in web pages. For instance, some ISPs began to tamper with HTTP traffic in transit, injecting DOM elements into HTML documents that added ISP's advertisements into pages visited by their customers [10, 30]. In a similar fashion, browser extensions started modifying pages to inject DOM elements in order to show ads to users without necessarily obtaining the user's prior consent. Ad injection has evolved to become a common form of unrequested third-party content injection on today's web [37].

These practices have several effects on both publishers and users. On one hand, ad injection diverts revenue from the publisher to the third party responsible for the ad injection. If advertisements are the primary source of income for a publisher, this can have a significant effect on their bottom line. If the injected ads contain or reference undesired content (e.g., adult or political topics), ad injection can also harm the reputation of the publisher from the user's perspective. If the content injection is also malicious in nature, the publisher's reputation can be further harmed in addition to exposing users to security risks due to malware, phishing, and other threats. Prior work has shown that users exposed to ad injection are more likely to be exposed to "malvertising" and traditional malware [46, 48]. Figure 1 gives an overview of ad injection's effect on the normal ad delivery process, while Figure 3 shows an instance of ad injection on amazon.com.

4

Extension	No. of Users	Injected Element
Adblock Plus	10,000,000+	`<iframe>`
Google Translate	6,000,000+	`<div>`
Tampermonkey	5,800,000+	`<img>`
Evernote Web Clipper	4,300,000+	`<iframe>`
Google Dictionary	3,000,000+	`<div>`

Table 1: Five popular Chrome extensions that modify web pages as part of their benign functionality.

2.3 Motivation

Recently, there have been efforts by browser vendors to remove ad-injecting extensions from their repositories [1]. Although semi-automated central approaches have been successful in identifying ad-injecting extensions, deceptive extensions can simply hide their ad injection behaviors during the short period of analysis time. In addition, finding web pages that trigger ad injection is a non-trivial task, and they can miss some ad-injecting extensions. Moreover, there are extensions that are not provided through the web stores, and users can get them from local marketplaces, which may not examined the extensions properly. Hence, we believe that there is a need for a protection tool to combat ad injection on the client side in addition to centralized examination by browser vendors.

Furthermore, automatically determining whether third-party content modification – such as that due to ad injection – should be allowed is not straightforward. Benign extensions extensively modify web pages as part of their normal functionality. To substantiate this, we examined five popular Chrome extensions as of the time of writing; these are listed in Table 1. Each of these extensions are available for all major browsers, and all modify web pages (e.g., inject elements) to implement their functionality. Therefore, automated approaches based on this criterion run a high risk of false positives when attempting to identify malicious or undesirable extensions.

Moreover, it is not enough to identify that advertisements, for instance, have been injected by a third party. This is because some users *might legitimately desire* the content that is being added to web pages by the extensions they install. To wit, it is primarily for this reason that a recent purge of extensions from the Chrome Web Store did not encompass the entirety of the extensions that were identified as suspicious in a previous study, as the third-party content modification could not be clearly considered as malicious [46]. Instead, we claim that *users themselves* are best positioned to make the determination as to whether third-party content modification is desired or not. An approach that proceeds from this observation would provide sufficient, easily comprehensible information to users in order to allow an informed choice as to whether content is desirable or should be blocked. It should be noted that defending against drive-by downloads and general malware is not the focus of this paper. Rather, the goal is to highlight injected ads to increase likelihood that user will make an informed choice to not click on them.

We envision that ORIGINTRACER could be used as a complementary approach to existing techniques such as central approaches used by browser vendors. Also, browser vendors can benefit from using our system in addition to end users to detect the con-

tent modifications by extensions in a more precise and reliable way. In the following sections, we present design and implementation of our system.

3 Web Content Provenance

In this section, we describe an in-browser approach for identifying third-party content modifications in web browsers. The approach adds *fine-grained provenance tracking* to the browser, at the level of individual DOM elements. Provenance information is used in two ways: *i)* to distinguish between content that originates from the web page publisher and content injected by an unassociated third party, and *ii)* to indicate *which* third party (e.g., extension) is responsible for content modifications using provenance indicators. By integrating the approach directly into the browser, we guarantee the trustworthiness of both the provenance information and the visual indicators. That is, as the browser is already part of the trusted computing base (TCB) in the web security model, we leverage this as the appropriate layer to compute precise, fine-grained provenance information. Similarly, the browser holds sufficient information to ensure that provenance indicators cannot be tampered with or occluded by malicious extensions. While we consider malicious or exploited browser plug-ins such as Flash Player outside our threat model, we note that modern browsers take great pains to isolate plug-ins in least privilege protection domains. We report separately on the implementation of the approach in Section 4.

In the following, we present our approach to tracking and propagating content provenance, and then discuss provenance indicators and remediation strategies.

3.1 Content Provenance

Web pages are composed of HTML that references resources such as stylesheets, scripts, images, plug-ins such as Flash objects, or even other web pages loaded inside frames. The document object model (DOM) is a natural structural representation of a web page that can be manipulated through a standard API, and serves as a suitable basis for provenance tracking. In particular, our system tracks the provenance of each element e contained in a DOM. Provenance for a DOM element is recorded as a set of labels $\ell \in \mathcal{P}(L)$, where the set of all labels L corresponds to a generalization of standard web origins to include extensions. That is, instead of the classic origin 3-tuple of $\langle \mathsf{scheme}, \mathsf{host}, \mathsf{port} \rangle$, we record

$$L = \langle S, I, P, X \rangle$$
$$S = \{\mathsf{scheme}\} \cup \{\text{``extension''}\}$$
$$I = \{\mathsf{host}\} \cup \{\mathsf{extension\text{-}identifier}\}$$
$$P = \{\mathsf{port}\} \cup \{\mathsf{null}\}$$
$$X = \{0, 1, 2, \ldots\}$$

In other words, a label is a 4-tuple that consists of a normal network scheme or extension, a network host or a unique extension identifier, a port or the special null value,
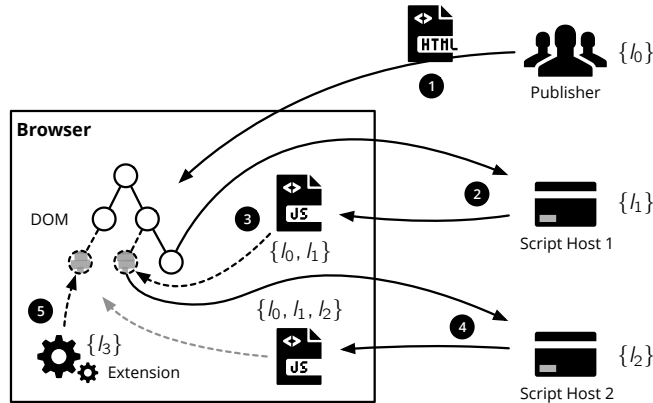
Fig. 2: Element-granularity provenance tracking. **(1)** Content loaded directly from the publisher is labeled with the publisher's origin, l_0. **(2)** An external script reference to origin l_1 is performed. **(3)** DOM modifications from l_1's script are labeled with the label set $\{l_0, l_1\}$. **(4)** Further external script loads and subsequent DOM modifications induce updated label sets – e.g., $\{l_0, l_1, l_2\}$. **(5)** A DOM modification that originates from an extension produces provenance label sets $\{l_0, l_1, l_2, l_3\}$ for the element.

and an index used to impose a global total order on labels as described below. While browsers use different extension identifiers, including randomly-generated identifiers, the exact representation used is unimportant so long as there is a one-to-one mapping between extensions and identifiers and their use is locally consistent within the browser. An overview of provenance tracking is depicted in Figure 2.

Static Publisher Provenance. Content provenance tracking begins with a web page load. As the DOM is parsed by the browser, each element is labeled with a singleton label set containing the origin of the publisher, $\{l_0\}$. Thus, static provenance tracking is straightforward and equivalent to the standard use of origins as a browser security context.

Dynamic Publisher Provenance. Content provenance becomes more interesting in the presence of dynamic code execution. As JavaScript can add, modify, and remove DOM elements in an arbitrary fashion using the DOM API exposed by the browser, it is necessary to track these modifications in terms of provenance labels.

New provenance labels are created from the publisher's label set $\{l_0\}$ as follows. Whenever an external script is referenced from the initial DOM resulting from the page load, a new label $l_i, i \in \{1, 2, \ldots\}$ is generated from the origin of the script. All subsequent DOM modifications that occur as a result of an external script loaded from the initial DOM are recorded as $\{l_0, l_i\}$. Successive external script loads follow the expected inductive label generation process – i.e., three successive external script loads from unique origins will result in a label set $\{l_0, l_i, l_j, l_k\}$. Finally, label sets contain unique elements such that consecutive external script loads from a previously accessed origin are not reflected in the label for subsequent DOM modifications. For instance, if the web page publisher loads a script from the publisher's origin, then any resulting

7

DOM modifications will have a provenance label set of $\{l_0\}$ instead of $\{l_0, l_0\}$. Content provenance is propagated for three generic classes of DOM operations: element insertion, modification, and deletion.

Element insertions produce an updated DOM that contains the new element labeled with the current label set, and potentially generates a new label set if the injected element is a script. Element modifications produce a DOM where the modified element's label set is merged with the current label set. Finally, element deletions simply remove the element from the DOM.

Extension Provenance. The third and final form of provenance tracking concerns content modifications due to DOM manipulations by extensions. In this case, provenance propagation follows the semantics for the above case of dynamic publisher provenance. Where these two cases differ, however, is in the provenance label initialization. While provenance label sets for content that originates, perhaps indirectly, from the web page publisher contains the publisher's origin label l_0, content that originates from an extension is rooted in a label set initialized with the *extension's* label. In particular, content modifications that originate from an extension *are not labeled* by the publisher's origin. An exception to this occurs when the extension, either directly or indirectly, subsequently loads scripts from the publisher, or modifies an existing element that originated from the publisher.

3.2 Content Provenance Indicators

With the fine-grained content provenance scheme described above, identifying the principal responsible for DOM modifications is straightforward. For each element, all that is required is to inspect its label set ℓ to check whether it contains the label of any extension.

A related, but separate, question is how best to relay this information to the user. In this design, several options are possible on a continuum from simply highlighting injected content without specific provenance information to reporting the full ordered provenance chain from the root to the most recent origin. The first option makes no use of the provenance chain, while the other end of the spectrum is likely to overwhelm most users with too much information, degrading the practical usefulness of provenance tracking. We suspect that a reasonable balance between these two extremes is a summarization of the full chain, for instance by reporting only the label of the corresponding extension.

Finally, if a user decides that the third-party content modification is unwanted, another design parameter is how to act upon this decision. Possible actions include blocking specific element modifications, removing the offending extension, or reporting its behavior to a central authority. We report on the specific design choices we made with respect to provenance indicators in the presentation of our implementation in Section 4.

4   OriginTracer

In this section, we present ORIGINTRACER, our prototype implementation for identifying and highlighting extension-based web page content modifications. We implemented

ORIGINTRACER as a set of modifications to the Chromium browser. In particular, we modified both Blink and the extension engine to track the provenance of content insertion, modification, and removal according to the semantics presented in Section 3. These modifications also implement provenance indicators for suspicious content that does not originate from the publisher. In total, our changes consist of approximately 900 SLOC for C++ and several lines of JavaScript[1]. In the following, we provide more detail on the integration of ORIGINTRACER into Chromium.

4.1 Tracking Publisher Provenance

A core component of ORIGINTRACER is responsible for introducing and propagating provenance label sets for DOM elements. In the following, we discuss the implementation of provenance tracking for publisher content.

Tracking Static Elements. As discussed in Section 3, provenance label sets for static DOM elements that comprise the HTML document sent by the publisher as part of the initial page load are equivalent to the publisher's web origin – in our notation, l_0. Therefore, minimal modifications to the HTML parser were necessary to introduce these element annotations, which is performed in an incremental fashion as the page is parsed.

Tracking Dynamic Elements. To track dynamic content modifications, this component of ORIGINTRACER must also monitor JavaScript execution. When a `script` tag is encountered during parsing of a page, Blink creates a new element and attaches it to the DOM. Then, Blink obtains the JavaScript code (fetching it from network in the case of remote script reference), submits the script to the V8 JavaScript engine for execution, and pauses the parsing process until the script execution is finished. During execution of the script, some new elements might be created dynamically and inserted into the DOM. According to the provenance semantics, these new elements inherit the label set of the script. In order to create new elements in JavaScript, one can *i)* use DOM APIs to create a new element and attach it to the web page's DOM, or *ii)* write HTML tags directly into the page.

In the first method, to create a new element object, a canonical example is to provide the tag name to the `createElement` function. Then, other attributes of the newly created element are set – e.g., after creating an element object for an `a` tag, an address must be provided for its `href` attribute. Finally, the new element should be attached to the DOM tree as a child using `appendChild` or `insertBefore` functions. In the second method, HTML is inserted directly into the web page using the functions such as `write` and `writeln`, or by modifying the `innerHTML` attribute. In cases where existing elements are modified (e.g., changing an image's `src` attribute), the element inherits the label set of the currently executing script as well. In order to have a complete mediation of all DOM modifications to Web page, `Node` class in Blink implementation was instrumented in order to assign provenance label sets for newly created or modified elements using the label set applied to the currently executing script.

Handling Events and Timers. An additional consideration for this ORIGINTRACER component is modifications to event handlers and timer registrations, as developers

---

[1] SLOC were measured using David Wheeler's SLOCCount [5].

make heavy use of event and timer callbacks in modern JavaScript. For instance, such callbacks are used to handle user interface events such as clicking on elements, hovering over elements, or to schedule code after a time interval has elapsed. In practice, this requires the registration of callback handlers via `addEventListener` API for events, and `setTimeout` and `setInterval` for timers. To mediate callbacks related to the addition and firing of events and timers, we slightly modified the `EventTarget` and `DOMTimer` classes in Blink, respectively. Specifically, we record the mapping between the running scripts and their registered callback functions, and then recover the responsible script for DOM modification during callback execution.

4.2 Tracking Extension Provenance

Chromium's extension engine is responsible for loading extensions, checking their permissions against those declared in the manifest file, injecting content scripts, dispatching background scripts and content scripts to the V8 script engine for execution, and providing a channel for communication between content scripts and background page.

Chromium extensions can manipulate the web page's content by injecting *content scripts* into the web page or using the `webRequest` API. Content scripts are JavaScript programs that can manipulate the web page using the shared DOM, communicate with external servers via `XMLHttpRequest`, invoke a limited set of `chrome.*` APIs, and interact with their owning extension's background pages. By using `webRequest`, extensions are also able to modify and block HTTP requests and responses in order to change the web page's DOM.

In this work, we only track content modifications by content scripts and leave identifying ad injection by `webRequest` for future engineering work. Prior work, however, has mentioned that only 5% of ad injection incidents occurred via `webRequest`; instead, Chrome extensions mostly rely on content scripts to inject advertisements [46]. Moreover, with modern websites becoming more complex, injecting stealthy advertisement into the page using `webRequest` is not a trivial task.

Tracking Content Script Injection and Execution. To track elements created or modified during the execution of content scripts, extension engine was modified to hook events corresponding to script injection and execution. Content scripts can be inserted into the web page using different methods. If a content script should be injected into every matched web page, it must be registered in the extension manifest file using the `content_scripts` field. By providing different options for this field, one can control when and where the content scripts be injected. Another method is programmatic injection, which is useful when content scripts should be injected in response to specific events (e.g., a user clicks the extension's browser action). With programmatic injection, content scripts can be injected using the `tabs.executeScript` API if the `tabs` permission is set in the manifest file. Either way, content scripts have a provenance label set initialized with the extension's label upon injection.

Handling Callback Functions. Chromium's extension engine provides a messaging API as a communication channel between background pages and content scripts. Background pages and content scripts can receive messages from each other by providing a callback function for the `onMessage` or `onRequest` events, and can send messages by invoking `sendMessage` or `sendRequest`. To track the registration
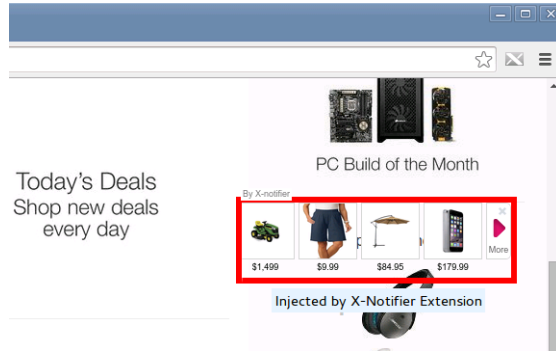
Fig. 3: An example of indicator for an injected advertisement on amazon.com.

and execution of callback functions, the `send_request` and `event` modules were slightly modified in the extension engine. Specifically, we added some code to map registered callbacks to their corresponding content scripts in order to find the extension responsible for DOM modification.

4.3 Content Provenance Indicators

Given DOM provenance information, ORIGINTRACER must first *i)* identify when suspicious content modifications – e.g., extension-based ad injection – has occurred, and additionally *ii)* communicate this information to the user in an easily comprehensible manner. To implement the first requirement, our prototype monitors for content modifications where a subtree of elements are annotated with label sets that contains a particular extension's label. This check can be performed efficiently by traversing the DOM and inspecting element label sets after a set of changes have been performed on the DOM.

There are several possible options to communicate content provenance as mentioned in Section 3. In our current prototype, provenance is indicated using a configurable border color of the root element of the suspicious DOM subtree. This border should be chosen to be visually distinct from the existing color palette of the web page. Finally, a tooltip indicating the root label is displayed when the user hovers their mouse over the DOM subtree. An example is shown in Figure 3. To implement these features, ORIGINTRACER modifies `style` and `title` attributes. In addition, since ORIGINTRACER highlights elements in an online fashion, it must delay the addition of highlighting until the element is attached to the page's DOM and is displayed. Therefore, modifications were made to the `ContainerNode` class that is responsible for attaching new elements to the DOM.

While we did not exhaustively explore the design space of content provenance indicators in this work (e.g., selective blocking of extension-based DOM modifications), we report on the usability of the prototype implementation in our evaluation.

11

5   Evaluation

In this section, we measure the effectiveness, usability, and performance of content provenance indicators using the ORIGINTRACER prototype. In particular, the questions we aim to answer with this evaluation are:

(Q1) How susceptible are users to injected content such as third-party advertisements? (§5.1)
(Q2) Do provenance indicators lead to a significant, measurable decrease in the likelihood of clicking on third-party content that originates from extensions? (§5.1)
(Q3) Are users likely to use the system during their normal web browsing? (§5.2)
(Q4) Does integration of the provenance tracking system significantly degrade the users' browsing experience and performance of the browser on a representative sample of websites? (§5.3)

Ethics Statement. As part of the evaluation, we performed two experiments involving users unaffiliated with the project as described below. Due to the potential risk to user confidentiality and privacy, we formulated an experimental protocol that was approved by our university's institutional review board (IRB). This protocol included safeguards designed to prevent exposing sensitive user data such as account names, passwords, personal addresses, and financial information, as well as to protect the anonymity of the study participants with respect to data storage and reporting. While users were not initially told the purpose of some of the experiments, all users were debriefed at the end of each trial as to the true purpose of the study.

5.1   Effectiveness of the Approach

Similar to prior work [13], we performed a user study to measure the effectiveness of content provenance in enabling users to more easily identify unwanted third-party content. However, we performed the user study with a significantly larger group of participants. The study population was composed of 80 students that represent a range of technical sophistication. We conducted an initial briefing prior to the experiments where we made it clear that we were interested in honest answers.

User Susceptibility to Ad Injection. The goal of the first phase of the experiment was to measure whether users were able to detect third-party content that was not intended for inclusion by the publishers of web pages presented to them. Users were divided into two equal sized groups of 40. In each group, users were first presented with three unmodified Chromium browsers, each of which had a separate ad-injecting extension installed: Auto Zoom, Alpha Finder, and X-Notifier for the first group, and Candy Zapper, uTorrent, and Gethoneybadger for the second group. These extensions were chosen because they exhibit a range of ad injection behaviors, from subtle injections that blend into the publisher's web page to very obvious pop-ups that are visually distinct from the publisher's content.

Using each browser, the participants were asked to visit three popular retail websites: Amazon, Walmart, and Alibaba. Each ad-injecting extension monitors for visits to these websites, and each injects three different types of advertisements into these
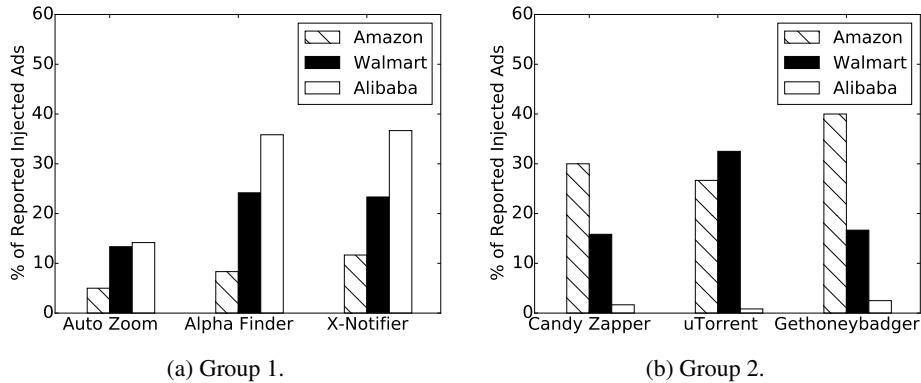
(a) Group 1.   (b) Group 2.

Fig. 4: Percentage of injected ads that are reported correctly by all the participants.

sites. For each website, we asked the participants to examine the page and tell us if they noticed any content in the page that did not belong to the website – in other words, whether any content did not seem to originate from the publisher. For each group, we aggregated the responses and presented the percentage of correctly reported ad injection incidents for each extension in Figure 4.

The results demonstrate that a significant number of Internet users often do not recognize when ad injection occurs in the wild, even when told to look for foreign content. For example, 34 participants did not recognize *any* injected ads out of the three that were added to Amazon website by Auto Zoom extension. Comparatively more users were able to identify ads injected by Alpha Finder and X-Notifier. We suspect the reason for this is because these extensions make use of pop-up advertisements that are easier to recognize as out-of-place. However, a significant number of users nevertheless failed to note these pop-up ads, and even after prompting stated that they thought these ads were part of the publisher's content. More generally, across all websites and extensions, many participants failed to identify any injected ads whatsoever.

We then asked each participant whether they would click on ads in general to measure the degree of trust that users put into the contents on the publisher's page. Specifically, we asked participants to rate the likelihood of clicking on ads on a scale from one to five, where one means that they would never click on an ad while five means that they would definitely click on an ad. We aggregated the responses and present the results in Figure 5a.

These results show that a significant number of users, roughly half, *would* click on advertisements that might not originate from the publisher, but that were instead injected by an extension. This demonstrates the effectiveness of ad injection as a mechanism for diverting revenue from publishers to extension authors. It also shows the potential effectiveness of malicious extensions in using content modifications to expose users to traditional malware.

Effectiveness of Content Provenance Indicators. After the first phase of the experiment, we briefly explained the purpose of ORIGINTRACER and content provenance to the participants. Then, for each participant in each group, we picked one of the three ad-injecting extensions in which, the participant did not detect most of the injected ads

(a) Susceptibility to ad injection.   (b) Ability to identify injected ads.   (c) Usability of content provenance.
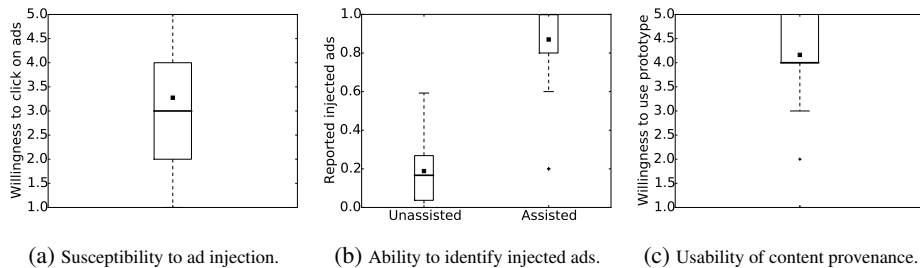
Fig. 5: User study results. For each boxplot, the box represents the boundaries of the first and third quartiles. The band within each box is the median, while the black square is the mean. The whiskers represent 1.5 IQR boundaries, and outliers are represented as a + symbol.

and installed it on a Chromium instance equipped with ORIGINTRACER. Then, each participant was asked to visit one of the three retail websites by his choice and identify third-party content modifications – i.e., injected ads – with the help of provenance indicators. The results are shown in Figure 5b, where unassisted identification is the aggregated number of reported ad injections without any assistance in the presence of three ad-injecting extensions across three retail websites, and assisted identification is the number of reported injected ads with the help of content provenance indicators. Results are normalized to $[0, 1]$.

These results clearly imply that users are more likely to recognize the presence of third-party content modifications using provenance indicators. To confirm statistical significance, we performed a hypothesis test where the null hypothesis is that provenance indicators do not assist in identifying third-party content modifications, while the alternative hypothesis is that provenance indicators do assist in identifying such content. Using a paired t-test, we obtain a p-value of 4.9199×10^{-7}, sufficient to reject the null hypothesis at a 1% significance level. The outliers in assisted identification are due to the fact that our ad highlighting technique was not identifiable by a small number of participants. We believe that using different visual highlighting techniques would make it easier for users to identify the injected ads.

Finally, we asked each participant how likely they would be to use the content provenance system in their daily web browsing. We asked participants to rate this likelihood on a scale from one to five, where one means they would never use the system and five means that they would always use it. The results are shown in Figure 5c, and indicate that most users would be willing to use a content provenance system. The reason behind the outliers is because a few of the participants stated that they do not need our system since they would not click on any advertisements. However, we note that it can be difficult to distinguish between advertisements and other legitimate content (e.g., products in retail sites) and, consequently, users might be lured into clicking on ad content injected by extensions.

Summary. From this user study, we draw several conclusions. First, we confirm that in many cases users are unable to distinguish injected third-party content from publisher content. We also show that because users place trust in publishers, they will often click on injected ads, and thus they tend to be susceptible to ad injection. Our data shows

that content provenance assists in helping users distinguish between trusted publisher content and injected third-party content that should not be trusted. Finally, we show that many users would be willing to use the system based on their experience in this study.

5.2 Usability

We conducted another experiment on a separate population of users to measure the usability of the ORIGINTRACER prototype. The user population was composed of 13 students with different technical background. We presented the participants with ORIGINTRACER integrated into Chromium 43, and asked them to browse the web for several hours, visiting any websites of their choice. For privacy reasons, however, we asked users to avoid browsing websites that require a login or that involve sensitive subject matter (e.g., adult or financial websites). In addition, for each user, we randomly selected 50 websites from the Alexa Top 500 that satisfy our user privacy constraints and asked the user to visit them. In particular, each participant was asked to browse at least three levels down from the home page and visit external links contained in each site. Finally, to gain some assurance that ORIGINTRACER would not break benign extensions, we configured the browser with the five high-profile extensions list in Table 1.

During the browsing session, the browser was modified to record the number of URLs visited. We also asked participants to record the number of pages in which they encountered one of two types of errors. Type I errors are those where the browser crashed, system error messages were displayed, pages would not load, or the website was completely unusable for some other reason. Type II errors include non-catastrophic errors that impact usability but did not preclude it – e.g., the page took an abnormally long time to load, or the appearance of the page was not as expected. We also asked users to report any broken functionality for the benign extensions described above as well.

Out of close to 2,000 URLs, two catastrophic errors and 27 non-catastrophic errors were encountered. However, we note that the majority of URLs rendered and executed correctly. In addition, none of the participants reported any broken extensions. We therefore conclude that the proposed approach is compatible with modern browsers and benign extensions, and further work would very likely allow the prototype to execute completely free of errors.

5.3 Performance

To measure the performance overhead of ORIGINTRACER, we configured both an unmodified Chromium browser and the prototype to automatically visit the Alexa Top 1K. The Alexa Top 1K covers many popular websites and is weighted towards being representative of the sites that people use most often. By using this test set, we ensured that each browser visited a broad spectrum of websites that include both static and dynamic content, and especially websites that make heavy use of third-party components and advertisements. Moreover, we configured both browser instances with the five benign extensions discussed in Section 2 that change the DOM to measure performance in the presence of extensions. A more detailed evaluation would analyze more pages on these

websites to garner a more realistic representation, but that is beyond the scope of the current work.

We built a crawler based on Selenium WebDriver [44] to automatically visit the entire list of websites and recorded the total elapsed time from the beginning of the browsing process until the entire list of websites was visited. Specifically, our crawler moves to the next website in the list when the current website is fully loaded, signified by the firing of the `onload` event. In order to account for fluctuations in browsing time due to network delays and the dynamic nature of advertisements, we repeated the experiment 10 times and measured the average elapsed time. The average elapsed time for browsing the home pages of the Alexa Top 1K websites measured in this way is 3,457 seconds for the unmodified browser and 3,821 seconds for ORIGINTRACER. Therefore, ORIGINTRACER incurred a 10.5% overhead on browsing time on average. We also measured the delay imposed by ORIGINTRACER on startup time by launching the browser 10 times and measuring the average launch time. ORIGINTRACER did not cause any measurable overhead on startup time.

While this overhead is not insignificant, we note that our user study in Section 5.2 indicates that many users would be willing to trade off actual perceived performance overhead against the security benefits provided by the system. Moreover, this prototype is just a proof-of-concept implementation of our system and there is still room for optimizing the implementation to decrease the page load time.

6 Related Work

6.1 Malicious Advertising

Substantial research on malicious advertisements has focused on isolation and containment [3, 15, 34]. Other approaches have focused on detecting drive-by downloads by employing the properties of HTTP redirections to identify malicious behavior [38, 45]. Dynamic analyses have also been used to detect drive-by downloads and web-hosted malware [11, 12, 36]. Li et al. [31] investigated the advertisement delivery process to detect malvertising by automatically generating detection rules. Web tripwires [43] were proposed to detect in-flight page changes performed by ISPs to inject advertisements.

6.2 Browser Extension Security

Browser extension security has recently become a hot topic. The Chromium extension framework substantially improved the ability of users to limit the amount of privilege conferred upon potentially vulnerable extensions [7], and follow-on work has studied the success of this approach [18, 33]. Other work has broadly studied malicious extensions that attempt to exfiltrate sensitive user data [32, 35]. For instance, Arjun et al. showed that many extensions in the Chrome Web Store are over-privileged for the actual services they purport to provide [21].

A recent line of work has focused on the problem of ad injection via browser extensions. Thomas et al. [46] proposed a detection methodology in which, they used a priori knowledge of a legitimate DOM structure to report the deviations from that structure

as potential ad injections. However, this approach is not purely client-side and requires cooperation from content publishers. Expector [48] inspects a browser extension and determines if it injects advertisements into websites. Hulk [28] is a dynamic analysis system that automatically detects Chrome extensions that perform certain types of malicious behaviors, including ad injection. WebEval [26] is an automatic system that considers an extension's behaviors, code, and author reputation to identify malicious extensions distributed through the Chrome Web Store.

In contrast, our work does not attempt to automatically classify extensions that engage in content modification as malicious or not, but rather focuses on enabling users to make informed decisions as to whether extensions engage in desirable behavior or not.

6.3 Provenance Tracking

A significant amount of work has examined the use of provenance in various contexts. For instance, one line of work has studied the collection of provenance information for generic applications up to entire systems [19, 24, 42]. However, to our knowledge, no system considers the provenance of fine-grained web content comprising the DOM. Provenance tracking is also related to information flow control (IFC), for which a considerable body of work exists at the operating system level [16, 29, 49], the language level [40, 9], as well as the web [20, 25]. In contrast to our work, IFC is focused more on enforcing principled security guarantees for new applications rather than tracking and indicating data provenance for existing ones.

Numerous systems have examined the use of dynamic taint analysis, a related concept to provenance. Some prior work [8, 17] focuses on tracking information flow within the browser, Sabre [14] detects whether extensions access sensitive information within the browser, and DSI enforcement [41] defends against XSS attacks by preserving the integrity of document structure in the browser. While there is certainly an overlap between dynamic taint analysis and provenance, taint analysis is most often focused on simple reachability between sources and sinks, while provenance is concerned with precisely tracking principals that influenced data.

Finally, there is a line of work that examines provenance on the web. Some prior work [22, 23, 39] concerns coarse-grained ontologies for describing the origins of data on the web, and does not consider provenance at a fine-grained scale within the browser. ESCUDO [27] only considers the principals that can be controlled by web applications, and it does not handle plug-ins and browser extensions. LeakTracker [47] performs principal-based tracking on web pages to study privacy violations related to JavaScript libraries, but it only tracks injection of scripts into the page, and does not provide any provenance information for other types of DOM elements. Excision [6] is the closest work to ours, which tracks inclusions of different resources in web pages and blocks inclusion of malicious resources by analyzing inclusion sequences on the page. Although the techniques are similar, they are used for different purposes. Excision discards the injection of DOM elements that do not reference remote content (e.g., `div`), and aside from source attributes, that does not track modifications to DOM elements. However, ORIGINTRACER identifies all types of DOM modification in the page, and instead of blocking content originating from extensions, it highlights them in the context of the web page.

7 Conclusion

In this paper, we introduced fine-grained web content provenance tracking and demonstrated its use for identifying unwanted third-party content such as injected advertisements. We evaluated a prototype implementation, a modified version of Chromium we call ORIGINTRACER, through a user study that demonstrated a statistically significant improvement in the ability of users to identify unwanted third-party content. Our performance evaluation shows a modest overhead on a large representative sample of popular websites, while our user experiments indicate that users are willing to trade off a slight decrease in performance for more insight into the sources of web content that they browse. We also performed a comprehensive study on the content modifications performed by ad-injecting extensions in the wild.

In future work, we plan to explore other uses of provenance on the web. Due to the highly interconnected structure of the web and the oftentimes obscure nature of its trust relationships, we believe that surfacing this information in the form of provenance is a generally useful capability, and can be applied in other novel ways in order to lead to safer and more informed web browsing. Finally, we plan to open source our prototype implementation in the hopes that it will be useful to the wider research community.

References

1. The ad injection economy. `http://googleonlinesecurity.blogspot.com/2015/05/new-research-ad-injection-economy.html`.
2. Adblock Plus. `https://adblockplus.org/`.
3. ADsafe. `http://www.adsafe.org/`.
4. Ghostery. `https://www.ghostery.com/en/`.
5. SLOCCount. `http://www.dwheeler.com/sloccount/`.
6. Sajjad Arshad, Amin Kharraz, and William Robertson. Include me out: In-browser detection of malicious third-party content inclusions. In *Financial Cryptography and Data Security (FC)*, 2016.
7. Adam Barth, Collin Jackson, Charles Reis, and The Google Chrome Team. The security architecture of the chromium browser. Technical report, 2008.
8. Lujo Bauer, Shaoying Cai, Limin Jia, Timothy Passaro, Michael Stroucken, and Yuan Tian. Run-time monitoring and formal analysis of information flows in Chromium. In *Network and Distributed System Security Symposium (NDSS)*, 2015.
9. Stephen Chong, K. Vikram, and Andrew C. Myers. SIF: Enforcing confidentiality and integrity in web applications. In *USENIX Security Symposium*, 2007.
10. 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/`, 2012.
11. Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *International World Wide Web Conference (WWW)*, 2010.
12. Andreas Dewald, Thorsten Holz, and Felix C. Freiling. ADSandbox: Sandboxing JavaScript to fight malicious websites. In *Symposium on Applied Computing (SAC)*, 2010.
13. Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2006.

14. Mohan Dhawan and Vinod Ganapathy. Analyzing information flow in JavaScript-based browser extensions. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

15. Xinshu Dong, Minh Tran, Zhenkai Liang, and Xuxian Jiang. AdSentry: Comprehensive and flexible confinement of JavaScript-based advertisements. In *Annual Computer Security Applications Conference (ACSAC)*, 2011.

16. Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazieres, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

17. Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. Dynamic spyware analysis. In *USENIX Annual Technical Conference (ATC)*, 2007.

18. Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In *USENIX Conference on Web Application Development (WebApps)*, 2011.

19. Ashish Gehani and Dawood Tariq. SPADE: Support for provenance auditing in distributed environments. In *International Middleware Conference*, 2012.

20. Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazieres, John C. Mitchell, and Alejandro Russo. Hails: Protecting data privacy in untrusted web applications. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

21. Arjun Guha, Matthew Fredrikson, Benjamin Livshits, and Nikhil Swamy. Verified security for browser extensions. In *IEEE Symposium on Security and Privacy (Oakland)*, 2011.

22. Andreas Harth, Axel Polleres, and Stefan Decker. Towards a social provenance model for the web. In *Workshop on Principles of Provenance (PrOPr)*, 2007.

23. Olaf Hartig. Provenance information in the web of data. In *Workshop on Linked Data on the Web (LDOW)*, 2009.

24. Ragib Hasan, Radu Sion, and Marianne Winslett. SPROV 2.0: A highly configurable platform-independent library for secure provenance. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

25. Boniface Hicks, Sandra Rueda, Dave King, Thomas Moyer, Joshua Schiffman, Yogesh Sreenivasan, Patrick McDaniel, and Trent Jaeger. An architecture for enforcing end-to-end access control over web applications. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, 2010.

26. Nav Jagpal, Eric Dingle, Jean Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *USENIX Security Symposium*, 2015.

27. Karthick Jayaraman, Wenliang Du, Balamurugan Rajagopalan, and Steve J. Chapin. ESCUDO: A fine-grained protection model for web browsers. In *30th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2010.

28. Alexandros Kapravelos, Chris Grier, Neha Chachra, Chris Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *USENIX Security Symposium*, 2014.

29. Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. In *Symposium on Operating Systems Principles (SOSP)*, 2007.

30. Greg Kumparak. Real evil: ISP inserted advertising. `http://techcrunch.com/2007/06/23/real-evil-isp-inserted-advertising/`, 2007.

31. Zhou Li, Kehuan Zhang, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

32. Zhuowei Li, XiaoFeng Wang, and Jong Youl Choi. SpyShield: Preserving privacy from spy add-ons. In *International Conference on Recent Advances in Intrusion Detection (RAID)*, 2007.

33. Lei Liu, Xinwen Zhang, Guanhua Yan, and Songqing Chen. Chrome extensions: Threat analysis and countermeasures. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

34. Mike Ter Louw, Karthik Thotta Ganesh, and V.N. Venkatakrishnan. AdJail: Practical enforcement of confidentiality and integrity policies on web advertisements. In *USENIX Security Symposium*, 2010.

35. Mike Ter Louw, Jin Soon Lim, and V. N. Venkatakrishnan. Enhancing web browser security against malware extensions. *Journal in Computer Virology*, 4(3):179–195, 2008.

36. Long Lu, Vinod Yegneswaran, Phillip Porras, and Wenke Lee. BLADE: An attack-agnostic approach for preventing drive-by malware infections. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

37. Ginny Marvin. Google study exposes "tangled web" of companies profiting from ad injection. `http://marketingland.com/ad-injector-study-google-127738`, 2015.

38. Hesham Mekky, Ruben Torres, Zhi Li Zhang, Sabyasachi Saha, and Antonio Nucci. Detecting malicious HTTP redirections using trees of user browsing activity. In *Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2014.

39. Luc Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2–3):99–241, February 2010.

40. Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Symposium on Principles of Programming Languages (POPL)*, 1999.

41. Yacin Nadji, Prateek Saxena, and Dawn Song. Document structure integrity: A robust basis for cross-site scripting defense. In *Network and Distributed System Security Symposium (NDSS)*, 2009.

42. Devin J. Pohly, Stephen McLaughlin, and Kevin Butler. Hi-Fi: Collecting high-fidelity whole-system provenance. In *Annual Computer Security Applications Conference (ACSAC)*, 2012.

43. 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.

44. Selenium Contributors. Selenium: Web browser automation. `http://www.seleniumhq.org/`.

45. Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

46. Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, Niels Provos, and Moheeb Abu Rajab. Ad injection at scale: Assessing deceptive advertisement modifications. In *IEEE Symposium on Security and Privacy (Oakland)*, 2015.

47. Minh Tran, Xinshu Dong, Zhenkai Liang, and Xuxian Jiang. Tracking the trackers: Fast and scalable dynamic analysis of web content for privacy violations. In *Proceedings of the 10th international conference on Applied Cryptography and Network Security (ACNS)*, pages 418–435, 2012.

48. Xinyu Xing, Wei Meng, Udi Weinsberg, Anmol Sheth, Byoungyoung Lee, Roberto Perdisci, and Wenke Lee. Unraveling the relationship between ad-injecting browser extensions and malvertising. In *International World Wide Web Conference (WWW)*, 2015.

49. Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazieres. Security distributed systems with information flow control. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.