Analyzing the Dangers Posed by Chrome Extensions

Lujo Bauer Shaoying Cai[†⋆] Limin Jia Timothy Passaro Yuan Tian
Carnegie Mellon University [†]Singapore Management University [⋆]Institute for Infocomm Research
{lbauer,liminjia,tpassaro,yt}@cmu.edu [†]shaoyingcai.2009@smu.edu.sg [⋆]cais@i2r.a-star.edu.sg

***Abstract*—A common characteristic of modern web browsers is that their functionality can be extended via third-party add-ons. In this paper we focus on *Chrome extensions*, to which the Chrome browser exports a rich API: extensions can potentially make network requests, access the local file system, get low-level information about running processes, etc. To guard against misuse, Chrome uses a permission system to curtail an extension's privileges. We demonstrate a series of attacks by which extensions can steal data, track user behavior, and collude to elevate their privileges. Although some attacks have previously been reported, we show that subtler versions can easily be devised that are less likely to be prevented by proposed defenses and can evade notice by the user. We quantify the potential danger of attacks by examining how many currently available extensions have sufficient privileges to carry them out. As many web sites do not employ defenses against such attacks, we examine how many popular web sites are vulnerable to each kind of attack. Our results show that a surprisingly large fraction of web sites is vulnerable to many attacks, and a large fraction of currently available extensions is potentially able to carry them out.**

I. Introduction

Web-based services are increasingly popular. Through web browsers, users can conveniently access a wide range of services such as email, cloud-based file sharing, banking, health-care, and shopping. While a user is interacting with such services, her sensitive, personal information—such as bank account numbers and passwords—is exposed to the browser and to scripts running on web pages. Furthermore, much functionality in modern browsers is achieved by installing powerful extensions and plug-ins that can access user resources (e.g., files, webcam) normally managed by the local operating system. These powerful add-ons to the browser, combined with already fragile browser security, further broaden browsers' attack surfaces. As a result, malicious add-ons—or innocent but buggy ones—can allow attackers to gain access to a wide range of private, sensitive data, and computer resources.

These dangers can lead to significant problems. Users and commercial organizations (e.g., banks) can suffer financial loss due to theft of financial information or identity theft. Loss of user confidence can cause decreased use of existing web-based services or reluctance to try new ones, potentially causing financial loss and stifling innovation.

Current browsers use permissions and Content Security Policies (CSP) to protect users' data and other browser components. For instance, a web page can use a CSP to specify the origin of scripts that it is willing to run. The user can grant the permissions to allow extensions to access some web sites, and constrain the behaviors of extensions by assigning them appropriate permissions. However, these mechanisms in general are not adequate to protect users' data.

In this paper, we focus on *Chrome extensions*, add-ons that extend the functionality of the Chrome browser. Chrome exports a rich API to extensions: they can potentially make network requests, access the local file system, get low-level information about running processes, etc. To guard against malicious extensions, Chrome uses a permission system to curtail an extension's privileges. We demonstrate a series of attacks by which extensions can steal data, track user behavior, and collude to elevate their privileges. Although some of the attacks have previously been reported, we show that subtler versions of the attacks can easily be devised that are both less likely to be prevented by proposed defenses and can evade being noticed by the user. For instance, we implemented an extension that steals users' sensitive data such as order history from ebay.com, even though the extension does not request the host permission to access content from ebay.com (see Section III-B for details).

Additionally, we quantify the potential danger of attacks by examining how many currently available extensions in the Chrome store (as well as how many of the most popular 1000 extensions) have sufficient privileges to carry out the attacks. We carry out this analysis by listing the extensions in the Chrome store by popularity[1], downloading all extensions, and parsing their manifests, which specify permissions. As many web sites employ defenses against such attacks, we also investigate how many popular web sites are vulnerable to each kind of attack. We manually examine the home pages of the global top 100 sites as reported by Alexa (www.alexa.com/topsites) for presence of specific defenses against the attacks. Our results show that a surprisingly large fraction of web sites is vulnerable to many attacks, and that a large fraction of currently available extensions is potentially able to carry them out. For instance, the above-mentioned attack on ebay.com could be launched by over 82% of the top-1000 popular extensions, as it requires few permissions. The attack affects any web page that allows itself to be placed in an iframe. A manual examination of Alexa's top-100 sites reveals that this attack can potentially steal data from over 56% of those sites.

The paper proceeds as follows. Section II reviews Chrome's security architecture as it pertains to extensions. Section III describes data-theft attacks that can be mounted by a single extension; Section IV focuses on tracking user behavior; and Section V examines privilege escalation attacks. Section VI discusses potential countermeasures. Related work is discussed in Section VII, and we conclude with Section VIII. Table I summarizes our attacks and analysis results. We have implemented the majority of described attacks.

[1]https://chrome.google.com/webstore/category/extensions?_sort=1. Unless otherwise noted, all measurements and experiments were done on 2013-12-15.

II. Background

Chrome's extension architecture is based on component isolation and privilege separation [7]. A Chrome extension is a zipped bundle of files—HTML, CSS, JavaScript, images, etc. [2]. An extension comprises components of three types: *content scripts* that directly interact with web pages; an *extension core* that interacts with browser; and an *optional native binary* that interacts with the OS.

The extension core becomes active when the browser starts or, if the extension has background permission, after a user logs into their computer. An extension can inject content scripts into web pages loaded by the browser; each page has its own instance of an extension's content scripts. Each content script runs in the same process as the web page into which it is injected. The extension core, of which there is only one instance per extension, and the extension's native binary (if any), each run in a separate process.

Chrome provides more than 40 APIs to extensions. Through these, extension cores can get real-time status of the browser, e.g., the list of tabs and installed or running extensions/apps; access and modify user's data, e.g., bookmarks and history; change browser settings, e.g., content settings and font settings; update browser components, e.g., uninstall an extension, launch an app, close a tab; monitor, hijack, or modify arbitrary web requests; and send messages to other extensions. A key security feature of Chrome's extension architecture is that the capabilities of components are constrained based both on their type and on permissions granted to them.

Content scripts have high risk to be exploited by malicious web sites, as they directly interact with web pages. Hence, among extension components, content scripts have the lowest privilege, and can use only the APIs provided to web pages, called browser APIs, which include `XMLHttpRequest`, `JSON`, and HTML5 APIs. Content scripts can only access the subset of Chrome APIs that supports messaging between an extension and its content scripts (`chrome.extension` API).

Chrome APIs, browser APIs, and access to web pages are guarded by permissions. An extension asks for permissions by declaring them in its manifest file. Permissions are of two types: host permissions and API permissions. Host permissions are a set of URLs, and specify into which pages an extension can inject content scripts. For example, if a password manager only has https://www.ebay.com host permission, then it cannot access any other web page. An extension core can only access Chrome extension APIs protected by permissions if it has the corresponding permissions in its manifest. Access to certain browser APIs and Chrome extension APIs is also constrained by host permissions. For example, if an extension does not have host permission http://www.google.com (or an encompassing permission like *://*.*), then it cannot make an `XMLHttpRequest` to http://www.google.com, or block a web request to http://www.google.com, even if it has API permissions webRequest and webRequestBlocking.

III. Theft and Forgery of User Data

Users' sensitive data such as usernames, passwords, credit card numbers, social security numbers, and date of birth are frequently communicated through web pages. As previously reported [25], extensions that are granted permissions to access the pages containing this data can easily steal it. In this section we demonstrate several variants of this general type of attack. Some of these variants are difficult for users to detect; others require fewer permissions than previously reported and may be more difficult to defeat through previously proposed defenses.

A. Abusing the http://*/* Host Permission

The most common type of permission that extensions are given is the permissions to inject content scripts into web pages opened by the browser. This permission enumerates the pages an extension is allowed to access. Commonly, an extension's content scripts are allowed to run on any page browsed by the user; the permission to do so is denoted http://*/*.[2] 58.3% of the top-1000 most popular Chrome extensions, and 38.0% of all Chrome extensions have this permission. These injected content scripts can read any content on the page, including data entered by the user, the browser (the built in form filler), or other extensions (e.g., password managers like LastPass[3]). In the basic scenario, which has been previously reported [25], when the user visits a web page, the malicious extension injects scripts into the page. Since they are running in the page's environment, these scripts have the ability to read from the DOM the password that the user enters. To successfully carry out this basic attack, the extension needs to be installed and active in the browser at the time when the user accesses the targeted page. Finding the desired information to be extracted in the DOM of the target page may be non-trivial and is often page-specific. More concerningly, similar attacks can be carried out even against pages that a user has not opened. We discuss this next.

Attack A1. Suppose that a malicious extension is granted the host permission http://*/*. When at least one page has been loaded by the browser, the extension becomes active. At this point, the extension has several methods by which it can cause a new web page to be loaded without any input from the user. For example, a content script can invoke `chrome.tabs.update`, which doesn't require any permission, to redirect a page to a new URL. If the browser or other extensions have saved auto-fill data for that URL, it will be entered into this page and available to the malicious extension.

Stealthy attacks using background tabs While such redirection in the active tab might be easily noticed, it can also be performed more stealthily, in tabs in the background.

There are at least two stealthy ways to open or redirect a tab to target web pages that do not require additional permissions beyond the http://*/* host permission (specifically, and perhaps surprisingly, the tab permission is not required). The first method is to redirect an inactive tab to the target web page; the extension can then steal sensitive information,

[2]Host permissions have several different forms, including http|https|*://*/*, non-wildcard versions of the same, and all_hosts. Additionally, content scripts could be allowed to execute in pages even when the extension core isn't allowed to access data from those pages directly. For simplicity of explanation, we conservatively write that an extension has http://*/* host permission when its permissions allow it to access any page *both* using HTTP and using HTTPS.

[3]www.lastpass.com

attack ID	summary	permissions needed	warning shown	% top-1000 exts affected	% exts affected	% downloads affected	% top-100 sites affected	stealthy	needs user	high bandwidth	standard defense	info vulnerable	easy to deploy	novelty
A1	steal page and auto-filled secrets	`http://*/*`	access to all web sites	58.3	38.0	73.3	100.0	◐	N	●	none	any	●	○[25]
A1-1	load victim page in iframe + steal secrets	`http://*/*`, `all_frames`	access to all web sites	58.3	38.0	73.3	56.0	●	N	●	`X-Frame-Options`, framebusting	any	●	○[25]
A1-2	strip `X-Frame-Options` + steal secrets	`http://*/*`, `webRequest-Blocking`	access to all web sites	10.2	4.2	27.5	100.0	●	N	●	none	any	●	◐[4]
A2-1	steal visible secrets w/o host permission	*some* host permission	access to some web sites	82.3	77.3	89.4	56.0	●	Y	●	`X-Frame-Options`, framebusting	iframeable, visible	◐	●[33]
A2-2	steal secrets w/o host permission	`activeTab`, `all_frames`	none	>1.3	>1.6	>0.6	56.0	●	Y	●	`X-Frame-Options`, framebusting	iframeable	◐	●
A3	forge user input / violate data integrity	*some* host permission	access to some web sites	82.3	77.3	89.4	100.0	◐	N	●	none	any	●	●
A4-1	track user behavior across devices	`http://*/*`	access to all websites	58.3	38.0	73.3	100.0	●	N	●	none	any	●	◐[28]
A4-2	track user interest via mouse movement	`http://*/*`	access to all websites	58.3	38.0	73.3	100.0	●	N	●	none	any	●	◐[19], [20], [21]
A4-3	track data input via keylogging	`http://*/*`	access to all websites	58.3	38.0	73.3	100.0	●	N	●	none	any	●	○[1]
A5-1	indirectly capture browsing history	`http://*/*`	access to all websites	58.3	38.0	73.3	100.0	●	N	●	none	any	●	●
A5-2	capture browsing history through timing	*some* host permission	access to some web sites	82.3	77.3	89.4	100.0	●	N	◐	none	any	●	○[17], [9], [5]
A6	process-monitoring-based spying	`process`	access to tabs and browsing activities	73.2	60.4	84.2	N/A	●	N	●	none	any	●	●
A7-1	collusion via explicit shared state	none†	none†	100.0	100.0	100.0	100.0	●†	N	●†	none	any	◐	●
A7-2	collusion via transient side channels	none	none	100.0	100.0	100.0	100.0	●†	N	●†	none	any	◐	●[30], [27]

TABLE I. SUMMARY OF ATTACKS WE DISCUSS. "WARNING SHOWN" INDICATES THE WARNING SHOWN TO THE USER; SOME PERMISSIONS DO NOT CAUSE AN ADDITIONAL WARNING TO BE SHOWN. THE "% [TOP-1000] EXTS AFFECTED" INDICATES THE PERCENTAGE OF (TOP-1000 MOST POPULAR) EXTENSIONS IN THE CHROME STORE THAT DISPLAY AN INSTALLATION-TIME OR RUN-TIME WARNING CONSISTENT WITH THE PERMISSIONS REQUIRED TO CARRY OUT THE ATTACK. SOME PERMISSIONS DO NOT CAUSE WARNINGS TO APPEAR, AND SOME PERMISSIONS SHOW THE SAME WARNING. WE WRITE >X TO INDICATE THAT AT LEAST X% EXTENSIONS ARE AFFECTED; ADDITIONAL APPLICATIONS CAN BE AFFECTED BECAUSE AN API CALL CAUSES THE SAME WARNING TO BE DISPLAYED, BUT WE DID NOT EXAMINE THE CODE OF EXTENSIONS. FOR ATTACKS THAT REQUIRE THE USER TO CLICK ON THE EXTENSION ICON, WE REPORT THE PERCENTAGE OF EXTENSIONS THAT REQUIRE THIS BEHAVIOR. "% DOWNLOADS" NORMALIZES "% EXTS AFFECTED" BY THE NUMBER OF DOWNLOADS OF EACH EXTENSION. DOWNLOAD NUMBERS FOR THE THREE MOST POPULAR EXTENSIONS ARE APPROXIMATE, SINCE THE CHROME STORE CEASES REPORTING PRECISE DOWNLOAD NUMBERS WHEN AN EXTENSION IS DOWNLOADED MORE THAN 10 MILLION TIMES. "% TOP-100 SITES AFFECTED" DESCRIBES THE NUMBER OF ALEXA GLOBAL TOP-100 SITES THAT ON MANUAL EXAMINATION APPEAR SUSCEPTIBLE TO THE ATTACK. "STEALTHY" SHOWS WHETHER THE ATTACK IS EASILY NOTICED BY THE USER: ● INDICATES ATTACKS THAT HAVE NO EFFECTS OBSERVABLE TO THE USER; ◐ MIGHT BE OBSERVED BY A VIGILANT USER. "NEEDS USER" SHOWS WHETHER USER INPUT IS REQUIRED FOR THE ATTACK TO START. "HIGH BANDWIDTH" SHOWS ● IF THE CHANNEL FOR THEFT/COLLUSION CAN MOVE MEGABYTES PER OPERATION (E.G., DIRECT MESSAGING), ◐ IF THE CHANNEL IS BUILT FROM OPERATIONS THAT MOVE HUNDREDS OF BYTES AT A TIME (E.G., STORING DATA IN A COOKIE), AND ○ IF INDIVIDUAL OPERATIONS TRANSFER ONLY ONE OR SEVERAL BITS AT A TIME (E.G., CPU TIMING). "NEW OR IMPROVED" SHOWS ● IF THE ATTACK HAS NOT PREVIOUSLY BEEN REPORTED, ◐ IF THE DESCRIBED ATTACK IS A MORE SOPHISTICATED VERSION OF AN EXISTING ATTACK, AND ○ IF IT HAS BEEN PREVIOUSLY REPORTED. SOME ATTACKS (A2-1 AND A7-2) ARE ANALOGOUS TO ATTACKS REPORTED ON OTHER PLATFORMS, SUCH AS ANDROID. OTHER COLUMNS ARE SELF-EXPLANATORY. † INDICATES THAT DIFFERENT VARIANTS OF THIS ATTACK CAN BE CARRIED OUT, SOME REQUIRING PERMISSIONS, AND ACHIEVING DIFFERENT BANDWIDTH AND LEVELS OF STEALTH (SEE TABLE II).

and afterwards redirect the tab to the original web page. More specifically, by calling `chrome.tabs.query(queryInfo)`, with `queryInfo`'s `active` flag set to `false`, an extension can get the list of inactive tabs. The query can be further restricted to tabs open in background windows (if the browser has several windows open), by setting `queryInfo`'s `currentWindow` field to `false`. Then, the extension can use `chrome.tabs.update` to redirect the tab. These tab API methods are not considered sensitive by Chrome, and so the extension does not have to claim the tab permission in its manifest. The only potentially observable visual evidence of this attack is that the tab icon will redraw when a different page is loaded. An alternative to using `chrome.tabs.query` to determine whether a tab is visible is to use the `windows` API, which can be used to determine which browser windows (if any) are currently focused, i.e., are at the foreground or have the pointer hovering over them, enabling an extension to launch attacks only when the user is using an application other than the browser. Using the `windows` API in this way does not require the extension to have any permissions.

To confirm the feasibility of the attack, we implemented a Chrome extension that successfully "stole" the username and password from the Facebook.com and PayPal.com accounts of one of the authors. The extension asks for permission to all web pages (http://*/*). Once active on a victim page, the extension waits for 0.5 seconds to give the password manager time to fill in the username and password fields. It then collects the information entered into those fields invoking the `document.getElementById` method.

Stealthy attack via iframes An even less noticeable way for extensions to access pages is to load them into iframes, which can be in background tabs, or fully transparent or of very small size, rendering them unnoticeable to the user.

ATTACK A1-1. Suppose that an extension's content script is running in some page. The extension can then modify the DOM of that page to create a new iframe (e.g., by executing `document.write("<iframe src=\"http://victim.com\"> </iframe>");`). The desired page is loaded in the iframe, and password-manager extensions or the browsers auto-fill functionality will automatically fill in any remembered content for victim.com. To read that content inside an iframe, an extension needs to have host permission to the iframed page as well as the `all_frames` option specified in its manifest. The addition of the `all_frames` option causes no additional warning to be shown to the user on installation. We have implemented an attack that successfully steals the account name and password from eBay's login page.

Of Alexa's top-100 web sites, 44% cannot be loaded in an iframe: 36.0% specify this using the `X-Frame-Options` header; 5.0% use framebusting; and 3% use other types of defenses against iframes. However, extensions with the webRequestBlocking permission can intercept and rewrite HTTP headers to strip the `X-Frame-Options` header, and thus make the pages that use this defense vulnerable.

ATTACK A1-2. We implemented an extension to demonstrate the feasibility of stripping `X-Frame-Options`. The extension has host permission to all web pages and the webRequestBlocking permission. It registers a listener for the `chrome.webRequest.onHeadersReceived` event, and uses it to set the `X-Frame-Options` header to `ALLOW` for pages being fetched by the browser. Using the extension, we successfully loaded yahoo.com, which uses `X-Frame-Options` to prevent framing, into an iframe, enabling attacks like A1-1.

B. Abusing the `captureVisibleTab` *Method*

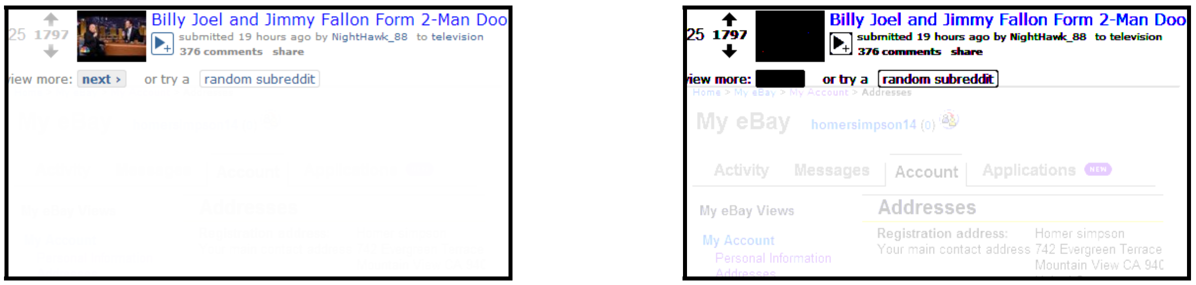
Extensions can also steal sensitive data from web sites for which they don't have host permissions by using the `captureVisibleTab` method of the `tab` API. This method enables an extension to capture a screenshot of all the content on the currently active tab, including any content rendered in iframes. This method is not protected by any permission, but is enabled only after the user has manually clicked on the extension's icon in Chrome's address bar (or toolbar or context menu). We next describe how this can be used to steal sensitive data without attracting users' attention.

ATTACK A2-1. Suppose that an extension has a content script running in a tab that *does not* contain the victim page (e.g., by having host permission to the specific page loaded in the tab). The extension can then create an iframe (see Section III-A) and load the victim page, allowing a password manager to automatically fill in any auto-fill content for that page.

The extension can then take a screenshot of the page on which it is running, and which now includes the auto-filled content. This method cannot be easily used to steal passwords or other information that is not visible in clear text, but a variety of sensitive information, including credit card numbers, date of birth, and usernames is typically shown in clear text. Some subtlety is required to ensure that the user does not notice this kind of manipulation of the page loaded in the active tab (the page into which the iframe is injected). The attacker can make the iframe almost transparent or open keep a very small window for the iframe and scroll the window to display the content. Another method to avoid being noticed by the user is to make the area of the iframe so small as to show only a single character at a time, and to move the field of view character by character until the entire secret has been captured by invoking `frame.contentWindow.scrollTo(xcoord,ycoord).`

As proof of concept, we developed an extension that steals a user's ebay.com account address by loading ebay.com inside a nearly transparent iframe. Figure 1(a) shows a screenshot of the host page (at reddit.com) with the eBay iframe loaded but invisible to the naked eye. Figure 1(b) shows the same image after adjusting the RGB curve of the layer; there, the account address is easily perceptible to the attacker.

Sometimes, sensitive information is available only after a user has logged in to a victim page (e.g., as for the eBay example above). Capturing such information requires that a malicious extensions mounts the attack after the user has logged in but before the user's session has expired. There are several ways for the extension to detect that a user has recently logged in, enabling it to attempt an attack only then. For extensions that have host permission to the desired page, noticing that a user has logged in is straightforward, since the extension can observe the loading of the login page. (Other permissions,


1797
Billy Joel and Jimmy Fallon Form 2-Man Doo
submitted 19 hours ago by NightHawk_88 to television
376 comments share
iew more: next › or try a random subreddit
My eBay
homersimpson14 (0)
Activity
Messages
Account
Applications
My eBay Views
Addresses
My Account
Personal Information

Registration address: Homer simpson
Your main contact address 742 Evergreen Terrace
Mountain View CA 940

(a) Without image post-processing, the injected iframe is invisible to the naked eye.

(b) After software manipulation, the attacker can easily extract sensitive information from the previously invisible iframe.

Fig. 1. Screenshots of a reddit.com page into which a malicious extension has injected an iframe that loads the user's eBay account information.

like webRequest or history, similarly straightforwardly give access to this information.) Some server-side defenses against session stealing may prevent this kind of access to specific pages [3], but are not uniformly implemented.

A limitation of this attack, beyond that it affects only data rendered in clear text, is that web pages that cannot be rendered in an iframe (44% of Alexa top-100 web sites) are not directly vulnerable. A second limitation is that the user must click on the extension's icon in the Chrome address bar for the extension to be activated, and that the user must already be visiting the victim page. 1.60% of extensions have the activeTab permission. Hey Girl [11], for example, is an innocent image find/replace extension; Search the Current Site [34] offers enhanced search functionality for the current page; Craigslist Peek [32] shows expanded content on craigslist pages at the user's request. Each of these extensions has the activeTab permission and host permission to only a limited number of pages; malicious extensions could be similarly disguised.

C. *Abusing the* activeTab *Permission*

The attacks described in Sections III-A–III-B relied on the ability of the malicious extension to inject its content script into some page within which an attack on a victim page is then mounted. Here we describe a method for extensions to gain access to such an intermediary page without having host permission access to it, using the activeTab permission. This permission allows an extension to inject content scripts into the DOM of the currently active tab regardless of whether the extension has host permission to the page, provided (similarly to the `captureVisibleTab` method) that the user has manually clicked on the extension's icon. Installing or activating an extension with the activeTab permission does not involve any warning dialogs being shown to the user.

Once an extension with the activeTab permission is activated, the extension can inject content scripts into the page loaded in the active tab. This gives it similar capabilities as extensions that have the http://*/* host permission.

ATTACK A2-2. A malicious extension with activeTab permission can inject a script to the page on which it is activated. The extension then can cause a victim page to be loaded in an iframe and steal its visible secrets as described in Section III-B.

A more powerful method of extracting secrets from the iframed victim page can be used if the malicious extension's manifest specifies the `all_frames` option (see Section III-A), which gives an extension the capability to steal secrets like passwords that are not visible in clear text. In practice, seemingly inconsistently with documentation, and highlighting the importance of implementing hypothetical attacks, this attack appears to be limited to iframes whose main page is from the same domain as the tab's top-level page.

We implemented an extension that pretends to offer enhanced printing functionality (similarly to the "Print Friendly & PDF" extension, which has been downloaded over 160,000 times [40]); when activated on an Amazon.com page, the extension loads the user's Amazon address book in an invisible iframe and obtains the user's shipping address.

D. *Forging User Input*

Extensions can not only steal sensitive data, but also forge web requests so that it appears that they come from the user.

ATTACK A3. This kind of misbehavior can be used to extend the ability of malicious extensions to cause harm in several ways. First, it extends their ability to gather information. An extension could even log in to victim site on behalf of the user, and then access information available only when the user is logged in (e.g., transaction history or addresses on PayPal.com). Whether and how a malicious extension can log in on behalf of the user depends on a specific page's implementation of the login process. As proof of concept, we extended the functionality of our extension that implements attack A1 to log in to Facebook.com and PayPal.com with the identity of the user whose username and password have been auto-filled by the browser or a password-manager extension.

Second, this kind of misbehavior also allows extensions to perform data integrity attacks. Examples include changing passwords filled in by the user or by a password manager to cause a denial of service (e.g., the above-described extension that logs into PayPal can be trivially modified to attempt to log in with an incorrect password, thus locking the user out); online banking operations can be changed and new ones can be manufactured (e.g., a user's request to his bank to pay a bill can be modified; a new request to transfer money can be created). As another example, an extension can modify the password on the login page, lock user out, and trick the user into resetting the password. Many sites ask the user to provide answers to pre-selected password-reset questions. The

extension can obtain these answers and leverage them to attack the user's other accounts or forge transaction requests.

IV. Tracking User Behavior

Extensions can engage in many malicious behaviors beyond those discussed in Section III. In this section we discuss a range of spying or tracking that extensions can carry out. Some of these behaviors are closely related to attacks discussed in Section III; others are enabled directly by the permissions granted to extensions, and require no trick or exploit. Such behaviors—even when in retrospect consistent with the permissions extensions are explicitly granted—are unlikely to fit users' mental models of what those permissions enable.

Cross-device tracking Tracking user browsing behavior is a common and lucrative endeavor, with companies vying to get the most complete picture of users' online behavior [28]. Many techniques have been developed to facilitate this, like identifying individual users by examining installed fonts and other system configuration settings, allowing ad networks to precisely track a single user's web surfing across sites.

Attack A4-1. A malicious extension with the http://*/* host permission that examined users' sensitive data could facilitate tracking not just across sites and sessions, which can already be easily done without client-side help, but also across multiple browsers and on shared-use computers (e.g., in libraries, and on shared family computers). If a single extension (or colluding extensions) is installed on all the browsers that some user uses, the extension(s) can identify the specific user, e.g., by recognizing that the same username has been used on different instances of Chrome to log in to the same site (e.g., if the user logs in to Facebook from her work, home, and friends' computers). The extension can then inject information that uniquely identifies the user (e.g., a hash of the user's Facebook username and password) into the user's web requests, or collude with a third party tracker to link that user's various browsing contexts.

Tracking mouse movement A number of efforts have refined approaches for resolving mouse cursor movement into an understanding of user intent [19], [20], [21]. Many parties might seek to exploit extensions' ability to perform such spying, from retailers and advertisers to national intelligence agencies that practice surveillance on a massive scale.

Attack A4-2. Extensions that have http://*/* host permission and inject their content scripts into web pages can access the state of the mouse pointer (e.g., by registering a listener for `document.onmousemove` events, the extension can track the position of the mouse on a page). This information, combined with access to DOM content, allows an extension to track what page content the mouse pointer is hovering over. This technique could be used, for example, to recognize which headlines or paragraphs on a news web site a user is particularly interested in; and even to probe the user for specific interests by injecting content (e.g., a specific news article or headline) to actively test the user's interest in it.

Keylogging An extension with the http://*/* host permission can capture keystrokes typed into any browser tab, enabling particularly pernicious tracking. For example, all of a user's web-based email communication could be monitored, regardless of email provider, and including even web-based anonymous remailers and text written in online document editors. Moreover, keylogging would also capture transient content that the user decided to delete, e.g., an incautiously composed email or tweet, or the email address of a contact to whom the user on second thought decided not to send an email.

Attack A4-3. As proof of concept, we developed an extension that captures keyboard input. The extension waits for a specific email address to be entered on any page; the extension then records other keystrokes typed into the page (presumably, the content of an email), and sends a copy of the collected keystrokes to a third-party web site. The extension has only the http://*/* permission, and we successfully used it to capture messages composed to one of the authors both in an anonymous remailer[4] and in Facebook.

A keylogging extension like this could continuously analyze collected text for key phrases or names, which could trigger an upload of captured data or initiate additional spying.

History sniffing Simpler tracking behaviors are also easily available to extensions with seemingly safe permissions.

Attack A5-1. Chrome allows only extensions that have the history permission to access the browsing history. However, most extensions have the http://*/* host permission, and any such extension that has been installed for any length of time can compile a detailed browsing history simply by recording the addresses of all the pages into which its content scripts have been injected (the content scripts can read the `document.URL` field and relay it to the core, which collates them). Extensions with the tabs permission can read URLs directly from the tab object URL fields, even without having host permission.

Attack A5-2. History sniffing can also be implemented more subtly. For example, it has been shown that page scripts can detect the difference in rendering time between links that have already been followed and those that have not [5], [9]. A malicious extension with host permissions to even a single site can mount the following attack. The extension first embeds in a page a link to a (fake) URL that could not have been visited, causing the browser to render the link text as an unvisited link. The extension then substitutes, without changing the link text, the fake URL for a real URL that the user may have visited. If the URL had been previously visited, the browser will need to repaint the link, which takes more time than if the link does not need to be repainted. This attack, as carried out by page scripts, has already been reported [5], [9]; we only observe that it can equally be carried out by extension content scripts.

Process monitoring Recent work showed that real-time network usage data (e.g., packet sizes) can be leveraged to track a user's location and, coupled with knowledge of application behavior or other public information, determine activity within applications (in an extreme case, the content of tweets) [42].

[4]http://gilc.org/speech/anonymous/remailer.html

ATTACK A6. An extension with the processes permission can use the `processes` API to access statistics about real-time CPU, memory, and network usage of the browser. Such an extension could use recently developed techniques [42] to gain significant insight into the behavior of users on pages whose content the extension is forbidden from accessing (according to its permissions). The processes API is currently available in the Canary early-adopters' release of Chrome; APIs introduced there usually transition into the public release.

V. EXTENSION PRIVILEGE ESCALATION

Many extension misbehaviors require that an extension has permission to access sensitive data (e.g., the system clipboard) as well as permission to access the network to leak the data to a third party. However, such extensions could raise a user's suspicion by requesting more permissions than seem necessary. To avoid raising suspicion, the functionality of such an extension could be split among multiple extensions. Each could have only an innocent-seeming set of privileges, but they could collude to carry out the same malicious behavior.

Consider, for example, the "Cyberx Password Generator" extension [29]. The extension's purpose is to generate passwords, which the user can copy-paste into the password fields of web pages. Since the extension does not have permissions that would appear to allow it to send data to others (e.g., http://*/*), it may seem to the non-expert user that extension must be safe to run. Similarly, a user may be willing to trust extensions that have host permission only to their own web sites (e.g., "Counter Strike Best Online Games Collection" [38], "Online SpongeBob Games" [39]), since they may appear to be unable to learn secrets that do not belong to their own site. However, such trust is misplaced, as we next show.

Colluding via direct messaging Extensions can communicate (regardless of their permissions) via inter-extension messaging. Using `chrome.runtime.onMessageExternal.addListener`, the receiving extension registers an event handler to wait for messages; the sender calls `chrome.runtime.sendMessage`. An extension A that has learned a secret (e.g., a password) can send the secret directly to another extension B, which is not able to learn the secret otherwise, but is able to send data (including the secret) to others.

Extension A can similarly collude with a page script (e.g., on a malicious or compromised site accessed by the user), even if the extension does not have host permission to access that web page. To do this, the extension lists the sites from which to allow connections under `externally_connectable` in its manifest. This does not incur any warning to the user.

Colluding via shared state Extensions can also collude via less obvious channels. Given appropriate permissions, extensions can access many types of shared state, including history, bookmarks, and the system clipboard. These can be used to communicate secrets with high bandwidth. Lower-bandwidth communication can be achieved via shared state like font settings and power settings (e.g., information can be communicated one bit at a time by changing the default font size). Table II enumerates such communication channels.

channel	*bandwidth*	*stealthy*	permission	*top-1000 exts (%)*	*exts (%)*	*downloaded exts (%)*
history	◐	◐	history	9.4	2.4	11.1
bookmarks	◐	◐	bookmarks	6.5	2.7	10.7
cookies	◐	◐	cookies	17.8	8.6	24.5
management	○	◐	management	12.6	4.2	13.8
clipboard	●	○	clipboardRead, clipboardWrite	1.0	0.9	0.5
downloads	◐	◐	downloads	0	0.2	< 0.1
contentSettings	○	◐	contentSettings	0.6	0.4	0.8
fontSettings	○	◐	fontSettings	0.1	0.1	< 0.1
message	●	●	none	100	100	100
URL redirect	◐	○	none	100	100	100

TABLE II. TYPES OF SHARED STATE THAT CAN BE USED BY COLLUDING EXTENSIONS. WHEN DESCRIBING BANDWIDTH, ● INDICATES CHANNELS THAT TRIVIALLY GENERALIZE TO ARBITRARY BANDWIDTH; ◐ CHANNELS CONSTRAINED BY FORMAT, BUT CAN TRANSFER 1000S OF BYTES PER MESSAGE; ○ CHANNELS WHERE MESSAGES TRANSFER INDIVIDUAL BITS. WHEN DESCRIBING STEALTH, ● MEANS A USER NOT ACTIVELY ENGAGED IN DEBUGGING WILL NOT NOTICE THE COLLUSION; ◐ THAT A CASUAL USER WILL NOT NOTICE REGARDLESS OF WHICH BROWSER INTERFACE SHE OPENS; ○ THAT A CASUAL USER COULD NOTICE UNUSUAL BEHAVIOR IF SHE PERFORMED A SPECIFIC COMMON ACTION (E.G., PASTING FROM THE CLIPBOARD) AT EXACTLY THE RIGHT MOMENT.

ATTACK A7-1. We developed two extensions that communicate arbitrary-length strings to each other via the `management` API (which requires the management permission, granted to 12.6% of top-1000 extensions and 4.2% of extensions overall). The extensions use the enabled/disabled status of an agreed-upon third extension as the communication channel. The sender communicates a 0 bit by disabling the third extension for 500 ms, and a 1 bit by disabling it for 1000 ms. (Smaller intervals are likely to work as well.) The receiver extension uses event listeners to observe when the extension is enabled or disabled; no delay is necessary between sending two bits.

Extensions from the same developer may find it particularly easy to collude. Among the 12,308 developers who published extensions in the Chrome Web Store, 14.9% published multiple extensions. We call the set of extensions authored by the same developer a group. About 70% of groups request different permissions for different extensions within the group. The group with the largest spread of permissions had 6 extensions, each requiring between 0 and 11 permissions [15]. The largest number of extensions published by one developer is 125. Interestingly, this developer, wips.com, offers to others the service of publishing and maintaining their extensions, giving it a perfect opportunity to create many colluding extensions.

Colluding via ephemeral state Extensions can also communicate (and collude) via standard timing channels (like CPU utilization [27]) and temporary shared state, e.g., the number and text of URL strings of open tabs. The number of open tabs is a particularly relevant covert channel, since an extension requires no permissions to open a new tab, close a tab that it opened, or count the number of tabs.

ATTACK A7-2. As proof of concept, we implemented two extensions that send one bit of information to each other in this

manner. One extension opens a tab by calling `window.open`; the other counts the number of tabs by first listing all the windows `chrome.windows.getAll`, and then adding the length of `window.tab` for each window. The extensions agree a priori on a time to communicate. If the number of tabs opened at the agreed-upon instant is greater than two, the receiver interprets that as 1; two or fewer is interpreted as 0. To avoid detection, extensions could communicate at times when the user is unlikely to be near the computer.

VI. Discussion of Defenses

In this section, we discuss existing and proposed defenses against attacks described in Sections III–V.

Content Security Policies (CSPs) Web sites can define CSPs to protect themselves from cross-site-scripting and data-injection attacks. Such policies specify from which origins different types of content (e.g., images, scripts) can be loaded; origins can also include extensions (i.e., a CSP can specify that scripts and other resources from a particular extension are allowed). CSPs can be used to defend against attacks described in Sections III and IV. A web site can specify a CSP that only allows scripts from vetted sources to be injected. However, Chrome allows an injected content script to load content from sources disallowed by the page's CSP into the content script's context. As a result, the content script can collude with downloaded scripts to carry out attacks.

Fine-grained permissions One proposed defense is to enforce fine-grained access control to APIs and DOM elements [25]. For instance, if accesses to sensitive data in the DOM are guarded by a finer-grained permission that is not included in the http://*.* host permission, then many of the attacks in Section III can be prevented. Similarly, if the host permission to a URL does not include permission to listen to mouse events, then attacks in Section IV can be prevented as well. Most of our attacks, however, like collusion attacks, cannot be prevented by a finer-grained permission system. Permission-based systems inherently suffer from privilege escalation and information leaks, no matter how fine-grained the permissions are (c.f. [23], [13]). The colluding attacks in Section V are examples of privilege escalation and information leakage.

Enforcing information-flow policies Several works proposed using static or dynamic taint analysis [6], [16] to ensure that sensitive data does not flow to unauthorized extensions, web pages, or remote servers. Some of the simple colluding attacks can be detected by such analyses.

Much work has investigated preventing information leakage via JavaScript in browsers (e.g., [14]). However, browser architectures include not only scripts from web pages, but also DOM elements, extensions, and plugins. Recent work has proposed building browsers ground-up to support rich information-flow policies [41].

The granularity at which data is protected (e.g., based on origin versus on arbitrarily fine-grained labels) and the granularity of the enforcement mechanism (e.g., tracking dataflow at variable level versus at extension-level granularity) affect the effectiveness of defenses. Protecting data based on origin and enforcing policy at component level seems to offer an attractive tradeoff between the strength of security guarantees and the efficiency of enforcement. Using such an approach, existing policies such as SOP, CSP, extension permissions, and their compositions (including policy conflicts) can potentially be expressed cleanly in one framework.

Increasing user awareness Mechanisms that allow users to understand the capabilities of extensions and make informed decisions about whether to install an extension can also be part of an effective defense against malicious extensions. For instance, permissions can be easily abused if users cannot understand the security and privacy implications of granting those permissions. Clear and intuitive explanations of the security implications of permissions are key to defending against extensions that request too many permisisons.

The Chrome web store could also facilitate the comparison of extensions based on their capabilities, and steer users towards ones that require fewer privileges but implement similar functionality. This could incentivize developers to adhere to least privilege principles to gain traction in the marketplace.

VII. Related Work

Academia and industry have devoted significant effort to improving the security and reliability of browsers [8], [7], [12], [18], [31], [35]. Most popular browsers, such as Chrome and Mozilla, enforce forms of component isolation and privilege separation. Even with improved architecture designs, new ways to exploit the users by launching attacks within the browser or compromising the browser are frequently reported [37], [17], [22]. Allowing browsers to be further extended by third-party extensions has brought a new set of security concerns [24], [36], [6], [26]. For the rest of this section, we focus on comparing our work with prior work on analyzing the security of Chrome extensions.

Recently, Carlini et al. performed a security review of 100 Chrome extensions, found 70 vulnerabilities, and demonstrated several attacks [10]. We consider a different attack model, and focus on attacks mounted by extensions and compromise the secrecy and integrity of users' data; whereas Carlini et al. focus on benign-but-buggy extensions and network and web attackers. Proposed defenses include banning insecure coding practices that commonly lead to vulnerabilities, e.g., the usage of HTTP scripts and inline scripts. Limiting the capabilities of extensions, in general, can mitigate attacks mounted by extensions. However, defenses proposed so far cannot prevent all attacks here without inhibiting useful extension functionality.

Most similar in spirit to our work is Liu et al.'s work that describes several attacks by extensions [25]. Our attacks A1, A1-1, and A1-2 are subtler variants of Liu et al.'s password sniffing-attack [25]. We additionally analyze Chrome extension APIs that do not cause a warning to be displayed to the user at install time, but can access sensitive data. Several attacks we discuss use Chrome APIs (e.g., Attack A2-1 and A6) that are outside the scope of that work. Liu et al. concluded that Chrome's inability to preventing these attacks is rooted in the violation of the principles of least privilege and privilege

separation, and proposed a set of countermeasures that enforce micro-privilege management and differentiate DOM elements based on the level of sensitivity of the data they contain. Most of our attacks, e.g., the collusion attacks (Section V), cannot be prevented such countermeasures.

VIII. Conclusion

We catalogued a number of ways in which extensions, individually or via collusion, can steal users' sensitive data, track their behavior, and forge their input. Though some of these attacks have been previously reported, our investigation shows that the dangers posed by malicious extensions are greater than commonly thought. We determine that large fractions of popular web sites are vulnerable, and many published extensions have sufficient privileges to carry out the attacks.

Acknowledgments

This research was supported in part by US Navy grant N000141310156; NSF grants 0917047, 1018211, and 1320470; and the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

[1] "Chrome extension keylogger," http://sourceforge.net/projects/extensionkeylog/, accessed: 2014-02-11.

[2] "Chrome extensions overview," developer.chrome.com/extensions/overview.html, accessed: 2013-12-05.

[3] "Cross-site request forgery (CSRF) prevention cheat sheet," https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet, accessed: 2013-12-15.

[4] "Getting around X-Frame-Options DENY in a Chrome extension," http://stackoverflow.com/questions/15532791, accessed: 2014-02-11.

[5] "Pixel perfect timing attacks with HTML5," contextis.com/files/Browser_Timing_Attacks.pdf, accessed: 2013-12-15.

[6] S. Bandhakavi, N. Tiku, W. Pittman, S. T. King, P. Madhusudan, and M. Winslett, "Vetting browser extensions for security vulnerabilities with VEX," *Commun. ACM*, vol. 54, no. 9, pp. 91–99, Sep. 2011.

[7] A. Barth, A. P. Felt, P. Saxena, and A. Boodman, "Protecting browsers from extension vulnerabilities," in *Proc. NDSS*, 2010.

[8] A. Barth, C. Jackson, C. Reis, and T. G. C. Team, "The security architecture of the Chromium browser," Tech. Rep., 2008. [Online]. Available: http://seclab.stanford.edu/websec/chromium/

[9] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *Proc. WWW*, 2007.

[10] N. Carlini, A. P. Felt, and D. Wagner, "An evaluation of the Google Chrome extension security architecture," in *Proc. USENIX Sec.*, 2012.

[11] K. Champagne, "Hey-Girl," chrome.google.com/webstore/detail/hey-girl/jcpmmhaffdebnmkjelaohgjmndeongip, accessed: 2013-12-02.

[12] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen, "A safety-oriented platform for web applications," in *Proc. IEEE S&P*, 2006.

[13] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy, "Privilege escalation attacks on Android," in *Proc. ISC*, 2010.

[14] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens, "FlowFox: A web browser with flexible and precise information flow control," in *Proc. ACM CCS*, 2012.

[15] disconnect.me, "Disconnect search, Facebook disconnect," chrome.google.com/webstore/search/disconnect, accessed 2013-12-16.

[16] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. Song, "Dynamic spyware analysis," in *Proc. USENIX ATC*, 2007.

[17] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *Proc. ACM CCS*, 2000.

[18] C. Grier, S. Tang, and S. T. King, "Designing and implementing the OP and OP2 web browsers," *ACM Trans. Web*, vol. 5, no. 2, pp. 11:1–11:35, May 2011.

[19] Q. Guo and E. Agichtein, "Towards predicting web searcher gaze position from mouse movements," in *Proc. CHI EA*, 2010.

[20] J. Huang, R. W. White, G. Buscher, and K. Wang, "Improving searcher models using mouse cursor activity," in *Proc. ACM SIGIR*, 2012.

[21] J. Huang, R. W. White, and S. Dumais, "No clicks, no problem: Using cursor movements to understand and improve search," in *Proc. ACM CHI*, 2011.

[22] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in JavaScript web applications," in *Proc. ACM CCS*, 2010.

[23] L. Jia, J. Aljuraidan, E. Fragkaki, L. Bauer, M. Stroucken, K. Fukushima, S. Kiyomoto, and Y. Miyake, "Run-time enforcement of information-flow properties on Android," in *Proc. ESORICS*, 2013.

[24] L. Liu, X. Zhang, and S. Chen, "Botnet with browser extensions," in *Proc. IEEE SocialCom*, 2011.

[25] L. Liu, X. Zhang, V. Inc, G. Yan, and S. Chen, "Chrome extensions: Threat analysis and countermeasures," in *Proc. NDSS*, 2012.

[26] R. S. Liverani and N. Freeman, "Abusing Firefox extensions," 2009.

[27] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, "Analysis of the communication between colluding applications on modern smartphones," in *Proc. ACSAC*, 2012.

[28] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *Proc. IEEE S&P*, 2013.

[29] realhacker.altervista.org, "Cyberx password generator," chrome.google.com/webstore/detail/cyberx-password-generator/mhdpbjioaheebp, accessed: 2013-12-02.

[30] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound Trojan for smartphones," in *Proc. NDSS*, 2011.

[31] S. Tang, H. Mai, and S. T. King, "Trust and protection in the Illinois browser operating system," in *Proc. OSDI*, 2010.

[32] tech4computer.wordpress.com, "Craigslist peek," chrome.google.com/webstore/detail/craigslist-peek/knpehhedikdgkbmhgagpcpcbclaidlmf, accessed: 2013-12-02.

[33] Y. Tian, Y.-C. Liu, A. Bhosale, L.-S. Huang, P. Tague, and C. Jackson, "All your screens are belong to us: Attacks exploiting the HTML5 screen sharing API," in *Proc. IEEE S&P*, 2014.

[34] tobias-schmidbauer.de, "Search the current site," chrome.google.com/webstore/detail/search-the-current-site/jliolpcnkmolaaecncdfeofombdekjcp, accessed: 2013-12-02.

[35] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter, "The multi-principal OS construction of the Gazelle web browser," in *Proc. USENIX Sec.*, 2009.

[36] J. Wang, X. Li, X. Liu, X. Dong, J. Wang, Z. Liang, and Z. Feng, "An empirical study of dangerous behaviors in Firefox extensions," in *Proc. Intl. Conf. Info. Sec.*, 2012.

[37] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, "Automated web patrol with strider honeymonkeys," in *Proc. NDSS*, 2006.

[38] www.flashgame90.com, "Counter strike best online games collection," chrome.google.com/webstore/detail/counter-strike-best-onlin/hbpkcodlmobmmbhdhfembofegbpghdnh?hl=en, accessed: 2013-12-15.

[39] ——, "Online SpongeBob games," chrome.google.com/webstore/detail/online-spongebob-games/blkommpkadaihnagjpjpjbhkgfoekldk?hl=en, accessed 2013-12-15.

[40] www.printfriendly.com, "Print friendly & PDF," chrome.google.com/webstore/detail/print-friendly-pdf/ohlencieiipommannpdfcmfdpjjmeolj, accessed: 2013-12-16.

[41] E. Z. Yang, D. Stefan, J. Mitchell, D. Mazières, P. Marchenko, and B. Karp, "Toward principled browser security," in *Proc. HotOS*, 2013.

[42] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt, "Identity, location, disease and more: inferring your secrets from Android public resources," in *Proc. CCS*, 2013.