EmPoWeb: Empowering Web Applications with Browser Extensions

Dolière Francis Somé

Université Côte d'Azur / Inria, France
doliere.some@inria.fr

***Abstract*—Browser extensions are third party programs, tightly integrated to browsers, where they execute with elevated privileges in order to provide users with additional functionalities. Unlike web applications, extensions are not subject to the Same Origin Policy (SOP) and therefore can read and write user data on any web application. They also have access to sensitive user information including browsing history, bookmarks, credentials (cookies) and list of installed extensions. They have access to a permanent storage in which they can store data as long as they are installed in the user's browser. They can trigger the download of arbitrary files and save them on the user's device.**

For security reasons, browser extensions and web applications are executed in separate contexts. Nonetheless, in all major browsers, extensions and web applications can interact by exchanging messages. Through these communication channels, a web application can exploit extension privileged capabilities and thereby access and exfiltrate sensitive user information.

In this work, we analyzed the communication interfaces exposed to web applications by Chrome, Firefox and Opera browser extensions. As a result, we identified many extensions that web applications can exploit to access privileged capabilities. Through extensions' APIS, web applications can bypass SOP and access user data on any other web application, access user credentials (cookies), browsing history, bookmarks, list of installed extensions, extensions storage, and download and save arbitrary files in the user's device.

Our results demonstrate that the communications between browser extensions and web applications pose serious security and privacy threats to browsers, web applications and more importantly to users. We discuss countermeasures and proposals, and believe that our study and in particular the tool we used to detect and exploit these threats, can be used as part of extensions review process by browser vendors to help them identify and fix the aforementioned problems in extensions.

I. Introduction

Browser extensions or addons are third party programs, that can extend the functionality of browsers and improve users' browsing experience. In this work, we focus on the WebExtensions API, a cross-browser extensions system compatible with major browsers including including Chrome, Firefox, Opera and Microsoft Edge [1], [2], [3], [4]. Tightly integrated to browsers, extensions have access to elevated browser APIs and features. For instance, they can make HTTP requests to get data from any web application server, including those where users are logged into, such as their mailing, banking, social network applications, etc. As a comparison, web applications are bound by the Same Origin Policy (SOP) [5] and cannot access other web applications data, unless they both implement mechanisms such as Cross-Origin Resource Sharing (CORS) [6].

Due to their privileged position in browsers, it is well understood that extensions pose serious security and privacy threats to user data [7], [8], [9], [10], [11], [12], [13]. Therefore, in order to limit extensions capabilities, a mandatory permission system requires that extensions explicitly declare the set of APIs they effectively need to access. Nonetheless studies have shown that extensions still require many permissions [14], [15], [16]. Extensions also go through a review process from browser vendors. But again studies have unveiled many malicious extensions circumventing the review process to exfiltrate sensitive user data [15], [17], [18]

Besides, a benign (non-malicious) extension can be buggy, allowing adversaries to exploit its vulnerabilities in order to get access to user sensitive data. One type of adversary that can exploit such vulnerabilities in extensions is the `web attacker` [9], [10], [13]. Indeed for security reasons, extensions and web applications execute in different and isolated contexts. Nonetheless, extensions have access to the DOM of webpages. Extensions and webpages can also set up communication channels to exchange data with one another using the `postMessage` API [19] for instance. Bandhakavi et al. [9] and Carlini et al. [10] found that a few extensions manipulate data extracted from webpages without sanitization, leading to privileged escalations, because such data can be influenced by a `web attacker`. Calzavara et al. [13] found that message passing APIs can be abused for privilege escalation, and formally characterized the privileges that be exploited by a `web attacker`.

Similarly to those studies, our threat model considers the web application as the attacker. Specifically, we seek to study at large scale, the security and privacy implications of message passing APIs [13] among extensions in the wild, whether there are extensions that can be exploited by web applications to access sensitive user information. For instance, an extension can set up an interface (register a listener) to receive from web applications, messages consisting of the URL of a resource (data) hosted by another web application. The extension then makes a request to fetch the data (since it can do so with any web application as it is not subject to the SOP) and returns the response to the web application that sent the URL. Hence, these communications channels are a way for a deliberately or accidentally vulnerable extension to indirectly give a web

application access to browser features and APIs that the web application is not directly allowed to access.

We built a static analyzer and applied it to the message passing interfaces exposed by Google Chrome, Firefox and Opera extensions to web applications. When the tool found that a privileged extension capability could potentially be exploited by web applications, the extension was flagged suspicious. By manually reviewing the code of suspicious extensions, we found that 197 of them (mostly on Chrome) can be exploited by web applications (attackers) to access elevated browser features and APIs and sensitive user information. The extensions we have found have vulnerabilities that can be exploited by web applications to (i) break the privilege separation between extensions and web applications and execute arbitrary code in extensions context, (ii) bypass the Same Origin Policy and access user data on other applications, (iii) read user cookies and use them to mount session hijacking attacks [20], (iv) access data such as user browsing history, bookmarks, list of installed extensions that besides violating user privacy can be used for tracking purposes [21], [22], [23], [24], (v) store and retrieve data from extensions persistent storage for tracking purposes and (vi) trigger the download of malicious software on the user device which execution can damage user data.

In summary, this paper shows the security and privacy threats associated with the interactions between browser extensions and web applications and makes the following contributions:

- We built a static analysis tool and analyzed extensions message passing interfaces at large-scale: 66,401, 9,391 and 2,523 extensions on Chrome, Firefox and Opera respectively. About 4.97%, 5.14% and 8.48% of Chrome, Firefox and Opera extensions respectively were flagged as suspicious.
- We identified 197 extensions that pose various security and privacy threats to browsers, web applications, and users. They can be exploited by web applications to bypass the SOP, read user cookies, browsing history, bookmarks, list of installed extensions, store and retrieve data from the extension storage, or download malicious files and store them on the user device.

We provide online a web page for navigating through the different results of this work, including the tool and videos demonstrating how we exploited the capabilities of some of the extensions.

In the beginning of October 2018, we reported our findings to the vendors who positively acknowledged the issues. For instance, Firefox has already removed all the reported extensions, and Opera all but 2. We are still discussing with Opera on how to fix the 2 extensions. With Chrome, we are also discussing with them on the actions to take. That notwithstanding, we argue that browser vendors need to review extensions more rigorously, in particular take into consideration the use of message passing interfaces in extensions. We think that tools such our static analyzer can help in identifying and fixing the security and privacy threats we have identified in this paper. We also discuss a few proposals on the current design of the message passing interfaces so as to mitigate the attacks that web applications can mount against extensions.

II. Background

A. Browser extensions capabilities

Standard web technologies (HTML, CSS and JavaScript) are used to develop extensions for major web browsers including Chrome, Opera, Firefox, and Microsoft Edge. Interestingly, their specific extensions APIs, [1], [3], [2], [4] are compatible with each other to some extent, making it easy to migrate extensions written for a specific browser to other browsers with just a few changes. This work focuses on these extensions, referred to as WebExtensions.

Extensions security considerations Extensions are third party software, that users install to alter their browsers behavior and improve their browsing experience. Tightly integrated to browsers, extensions have access to privileged browser features, making them interesting targets for attackers. Hence, to limit the harm that attackers could cause if they take control of an extension, the APIs that extensions effectively have access to are only those for which they have explicitly requested the related permission in their `manifest.json` file. Listing 1 shows an example of manifest file and the `permissions` (features and APIs) the extension will be granted at runtime.

```
{
    "permissions": [
      "<all_urls>",
      "storage",
      "management",
      "cookies"
      "history",
      "bookmarks",
      "downloads",
    ]
}
```

Listing 1: Permissions declaration in a manifest file

These are only a subset of all the capabilities provided by browsers to extensions. When installed, this extension will be granted full access (read/write) to data on any web application, thanks to its `host` permission `<all_urls>`. This implies that if the user is logged into a web application (such as mailing, banking, social networks), the extension also has access to the user's private data on that application. The rest of the permissions read straightforwardly. The `storage` permission allows the extension to store and retrieve data in the browser as long as it is installed. The permissions `management`, `cookies`, `history`, and `bookmarks` give the extension access to the list of installed extensions, web applications cookies, user browsing history and bookmarks respectively. With the `downloads` permission, the extension can download and save arbitrary files in the user device. At runtime, those APIs (and any extension-specific API in general) are all accessible via the `chrome` object in Chrome and Opera browsers, and via `browser` object in Firefox and

Microsoft Edge. To ease the readability of this work, we often omit the `chrome` and `browser` from extensions APIs names.

Architecture

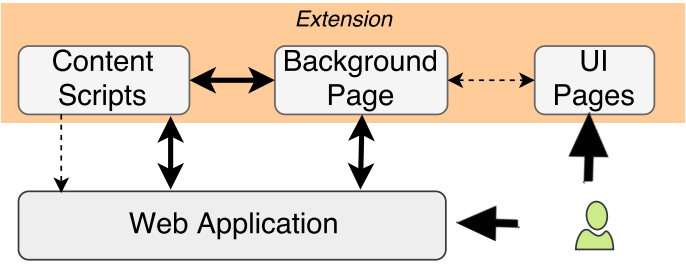

Fig. 1: Browser extensions architecture - Communications with web applications

Extensions can be divided in three main parts, as shown in Figure 1. The background page is the main part of the extension. It has full access to all the capabilities of the extension. Users interact with the extension through UI pages (i.e. UI elements, options and setting pages), in order to enable or disable it, or customize its behavior. UI pages also have access to the full capabilities of the extension. Content scripts are injected by extensions to run along web applications. Even though they are not granted access to all the extension capabilities, they can directly use the `host` and `storage` permissions to access user data on any web application or to store and retrieve data from the extension storage. Content scripts can also manipulate the DOM of webpages [25] and inject content in them. On Chrome and Opera, each extension is assigned a permanent unique identifier, which is the same for all users of the extension. Firefox however generates a random identifier for the extension, per user browser [26].

B. Interactions

Background and UI pages have direct access to each other's execution contexts [27], but content scripts execute in a separate context. Web applications run in yet other separate execution contexts. Nonetheless, content scripts have direct access to web applications localStorage, DOM, and execution context, where they can inject and execute arbitrary scripts.

Even though content scripts, background pages and web applications run in separate execution contexts, they can establish communication channels to exchange messages with one another [28], [29] as shown in Figure 1. We describe below the APIs for sending and receiving (listening for) messages between the content scripts, background pages and web applications.

Content scripts and background pages There are two types of communication channels: one-time and long-lived channels. One-time channels are opened to send a message and are closed after the response is received. Long-lived channels, connections or ports, are maintained open to exchange multiple messages. A port can have a name in order to distinguish it from other long-lived channels.

For one-time messages, content scripts use the `runtime.sendMessage` API to send messages to background pages. Similarly, background pages employ the `tabs.sendMessage` API to send messages to content scripts. For receiving messages, both components can invoke the `runtime.onMessage.addListener` API.

Similarly, `runtime.onConnect.addListener` and `runtime.connect` are used to establish long-term communications between background pages and content scripts.

Web applications and content scripts Exchanges between web applications and content scripts are achieved with the Cross-Origin Communications API [19]: `postMessage` is used for sending messages, and `onmessage` or `addEventListener` to receive messages. Below is a listing which shows how messages are sent and received between web applications and content scripts.

```
  // Send and receive
postMessage("Hello Extension", "*");
addEventListener("message", function(event){
  Received_response = event.data;
});
  // Receive and Reply
addEventListener("message", function(event){
  Received_message = event.data;
  postMessage("Hello Web Application", "*")
});
```

In this example, the web application sends the message *Hello Extension* to the content script, which receives and writes it in the variable `Received_message`. Then it replies with *Hello Web application*, which the web application receives and saves in the variable `Received_response`.

Web applications and background pages On Chrome and Opera, web applications can also directly communicate with extensions background pages. To do so, extensions have to declare in their `manifest.json` file, using the `externally_connectable` key, the list of web applications, where communication with the background page is allowed. For security reasons, one cannot use wildcard (for instance *) to allow communications between the background pages and all web applications. Additionally, communications can only be initiated by web applications.

The `runtime.sendMessage` and `runtime.connect` APIs are exposed to web applications in Chrome and Opera, and can be used to send one-time messages or establish long-term connections with background pages. Conversely, the APIs `runtime.onMessageExternal.addListener` and `runtime.onConnectExternal.addListener` can be used in the background page, to receive and reply to messages sent by web applications. Below is an example of how to send a message from a web application to the background page of an extension which unique identifier is `ExtensionID`.

```
  // Send and Receive
chrome.runtime.sendMessage(ExtensionID, "Hello
    Extension", function(response){
  Received_response = response;
});
  // Recieve and Reply
chrome.runtime.onMessageExternal.addListener(
    function(message, sender, sendResponse){
  Received_message = message;
```

```
  sendResponse("Hello Web application");
})
```

The application sends *Hello Extension* to the background page which replies with *Hello Web application*.

C. Threat models

An attacker is a script that is present in a web application currently running in the user browser. The script either belongs to the web application or to a third party. The goal of the attacker is to interact with installed extensions, in order to access user sensitive information. It relies on extensions whose privileged capabilities can be exploited via an exchange of messages with scripts in the web application. We consider the following security and privacy threats posed by extensions.

1) **Execute code**: these are extensions that can be exploited by the attacker to execute arbitrary codes in the extension context. Executing code in the background page gives the attacker access to all the capabilities of the extension. In content scripts, the attacker can bypass SOP by making cross-origin AJAX requests, and use the extension permanent storage for tracking purposes.
2) **Bypass SOP**: in this case, an attacker can exploit the capability of the extension to make cross-origin requests without being restricted by the Same Origin Policy.
3) **Read Cookies**: the attacker can read the user cookies and use them to mount session hijacking attacks, access user data and take actions on her behalf.
4) **Trigger Downloads**: the attacker exploits extensions to trigger the download of arbitrary malicious files (software) and saves them on the user's device without requiring any action from the user. If the user inadvertently runs such software, the attacker takes control of her device and performs malicious actions.
5) **Read browsing history, bookmarks and list of installed extensions**: these information reveal the user interests and habits and can be used by the attacker for tracking purposes, or to serve targeted and personalized advertisement.
6) **Store data**: the attacker can store and retrieve information in the extension storage. This can be used for tracking purposes, even though users clear web applications storages.

For the sake of simplicity, we often refer to the attacker as the web application in which it runs.

III. Methodology

We built a static analyzer that detects suspicious communications enabled by extensions with web applications. To identify extensions that are potentially concerned with the security and privacy threats identified in the previous section, we focus on 78,315 extensions from Chrome, Firefox and Opera browsers. Then we manually reviewed the code of the extensions to precisely validate the results of the static analyzer, and more importantly to construct the signatures of the messages that have to be exchanged with extensions to successfully exploit their capabilities. Figure 2 shows the analysis process.

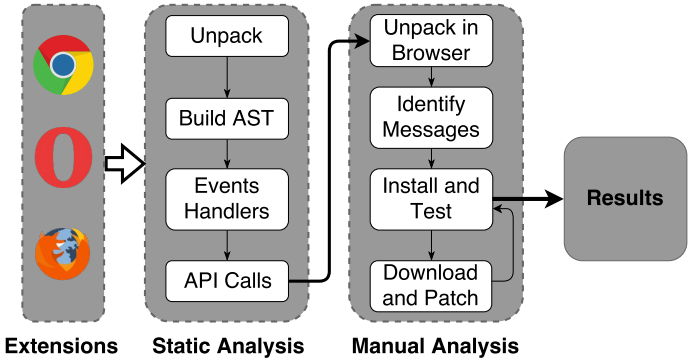

Fig. 2: Methodology - static and manual analysis

A. Static analysis

The goal of the static analyzer is to report only extensions that potentially pose a security and privacy threat, in order to reduce false positives as much as possible, and reduce the burden of the manual analysis. It has been fully written in JavaScript, using various Node.js packages. We used Esprima [30] and Recast [31], for parsing and manipulating JavaScript abstract syntax trees (AST), and Jsdom [32] for parsing HTML.

Unpack extensions and gather scripts We crawled extensions using SlimerJS Browser Automation tool [33]. In the extension `manifest.json` file, background pages are either declared by a set of scripts files, or an HTML file, which further includes the scripts of the background page. UI pages are built as HTML pages, and also indicated in `manifest.json` file. The Jsdom HTML parser served here to extract scripts embedded in background as well as UI pages. Static content scripts are directly declared in the `manifest.json` file. Background and UI pages can further dynamically inject content scripts in web applications, by calling the `tabs.executeScript` API. Those were also extracted by analyzing the AST of background and UI pages scripts, and analyzed as other content scripts.

Parse scripts and build AST Scripts were parsed with Esprima, resulting in an AST [34], which contains all JavaScript constructs used in content scripts, background and UI pages scripts. Almost everything in JavaScript is an object [35]. To ease manipulation of the AST, the following additional actions were taken to build three indexed tables of assignments to variables and object properties (`assignments`), function definitions/expressions and object methods (`functions`), and finally functions and object methods invocations (`calls`). Basically, those are key/value pairs, in which the keys in the tables corresponded to the names of variables, object properties and functions. Each entry was then associated with a list of all possible values it could resolve to. For assignments, the values were all expressions assigned to a variable or object. For function definitions and object methods,

the values were the parameters and body of the function. Finally, for function calls, the values associated to their names in the indexed table were their invocation arguments. The static analyzer successfully handled functions defined using the `bind` method, and functions invoked using the `call` or `apply` methods.

Event handlers of page messages APIs For each message listener (See Section II) in content scripts, background and UI pages, we first looked up the indexed table of function invocations (`calls`) to search whether the extension registered listeners for messages from the web applications (a call to `addEventListener` API for instance in content scripts). In browser contexts, all JavaScript objects are properties of a global object named `window`. Different aliases, `this, self, global`, are sometimes used to refer to the `window` object [36], [37]. JavaScript object properties can be accessed using the dot and the array or bracket notations [38]. For the sake of simplicity, we considered the dot notation and the bracket notation when the property name was a literal (a string). Considering the global object names (`window, top, self, this`), and JavaScript dot and bracket property accesses, we generated the different ways an API can be invoked. For instance, `addEventListener` can be called in 9 different ways `addEventListener, window.addEventListener, window["addEventListener"]` and others. In general, we consider that an object could be accessed in 9 different ways, its properties in 18 different ways, the properties of its properties in 36 ways and so forth. When we found an invocation to communications APIs in content scripts, background and UI pages, we extracted their arguments and resolved them as follows.

For `addEventListener`, the first argument should be the literal `message`, and the second argument a function. Otherwise, we use the indexed table of `assignments` and `functions` to resolve them to the literal `message` and a function respectively. Resolving an argument simply consist in checking whether the indexed table has an entry which key matches the argument name, and further checking whether any of its associated values resolve to the type and value we expect the argument to have. For `addEventListener`, we expect the first argument to be a `Literal` and have the value `message`. Its second argument is expected to be a function. We follow the same process to extract all message handlers (listeners) in content scripts, background and UI pages.

Sensitive APIs Calls The handlers (functions) of web applications messages in extensions are parsed to extract all their constructs. If the handlers further call other functions, those functions are looked up using the indexed table, and their bodies parsed to also extract their constructs. Finally, the constructs are analyzed to decide whether the extension potentially poses any of the security and privacy threats considered in this work.

- An extension is flagged as potentially executing arbitrary code sent from web applications if it invokes functions like `eval` (in any part of the extension) or `tabs.executeScript` (in background and UI pages).
- An extension is flagged as potentially allowing web applications to bypass SOP, if its constructs include APIs that can be used to make AJAX calls. This includes the creation of new `XMLHttpRequest` objects, calls to `fetch` API, or any AJAX specific API provided by popular third party libraries such jQuery and AngularJS (`$.get, $.ajax, $.post, $http.get, $http.post`).
- If the constructs include calls to `storage` API such as `storage.local.set, storage.local.get, storage.sync.set, storage.sync.get`, then the extension is flagged as potentially storing/retrieving data for web applications.
- An extension is considered as potentially leaking user cookies, history, bookmarks, and list of extensions to web applications if either of the following invocations were found in their message handlers constructs: `cookies.getAll, history.search, history.getVisits, bookmarks.getTree, management.getAll`, and related APIs.
- Finally, an extension is considered as probably allowing web applications to download and save files in the user computer (device) if their messages event handlers constructs include invocation to `downloads.download`.

It is worth mentioning the case of content scripts forwarding messages to background pages. When this is the case, the constructs of content scripts messages handlers in the background pages are also analyzed, looking for calls to any API which potentially poses security and privacy threats. In fact, content scripts only have access to the `host` and `storage` capabilities. When they need access to more capabilities, they can send messages to the background pages which may then give them access to the related capability. Content scripts can forward messages they receive from web applications, to the background page. The latter handles the message and responds to the content scripts which in turn respond to the application. This is particularly true in Firefox which does not allow direct communications between web applications and background pages. Nonetheless, we have observed many content scripts forwarding messages to background pages, even to access APIs they can directly use from their own context.

B. Manual Analysis

Recall that for each suspicious extension, the tool reports precisely (i) the component(s) (content scripts, background or UI pages) and the file or set of files to analyse, (iii) the name of the message passing interfaces registered (iv) the handlers of the message passing interfaces and other functions that are called from those handlers (v) and finally, the sensitive APIs that are triggered in the handler and its called functions. So the goal of the manual analysis was to confirm the suspicion of the static analyzer, build and test the precise signatures of messages that had to be sent by web applications to exploit extensions capabilities. Following is how we typically manually review an extension.

Unpack in Browser First, we download and unpack the extension code directly in a browser, using the CRX Extension Viewer [39][1]. We locate the files to be analyzed, according to the concerned component(s). In the particular case of content scripts, the tool reports the precise index in the content scripts array containing the file or set of files that has to be analyzed [2].

Identify Messages This step is concerned with building the payloads or signature of messages that can be sent to the extension to exploit is capabilities. In each file, we search for the message passing interfaces, their handlers and the functions they invoke. In those handlers and functions, we look for the sensitive APIs that are triggered and more importantly, the parts of the received messages that trigger calls to the extensions sensitive APIs. To build the payloads, we carefully inspect how objects received as parameters in the message passing interfaces handlers (and related functions) are manipulated, which properties of the objects are accessed, the names of the properties, and their values. This gives us the precise signature of the messages. Two situations arise here. Either we found that the signature of the messages are predefined by the extension, in which case they cannot be influenced by the attacker (See Section V-H for more details), and we stop the analysis and consider the extension as a false positive. Otherwise, we continue the analysis by installing and interacting with the extension by sending messages according to the signature that we have found.

Install and Test – Download and Patch The final step consists in mounting the exploits against the vulnerable extension with the payloads built in the previous step. Those interactions are done from the Browser console [40], after we navigate to the websites in which the extension injects its files. Sometimes, those websites require users to be logged in. We would create accounts and navigate to them. In rare cases, we could not create accounts or the websites were not responding. So we download the extension, and modify its permissions in order to make it inject files in websites that are accessible (i.e. http://localhost/), from were we mount the attack. During the tests, we use the browser debugger, set breakpoints in the extension codes in order to track the propagation of messages and calls in the extension code.

Time taken to manually analyze extensions It is rather difficult to precisely evaluate how much time it takes to manually analyze an extension. Indeed, this work went through 3 phases. We did a first crawl of extensions in the middle of November 2017 and run our static analyzer to test it. It took around 2 months and half (till the beginning of February 2018) to come with a mature analyzer. But during that phase, we had already discovered almost 87% of the extensions reported in this paper. Then, from the beginning of February 2018 we analyzed all the extensions again in around 4 weeks. Finally, in mid of May 2018, we did a new crawl and analysis, and it took around 10 days to vet the extensions. Again, for most of the extensions, we had already tested them, built the signatures of the messages, so analyzing them again consisted only in checking that they were still exploitable.

So overall, what we observed is that during the phase we implemented and tested the static analyzer, manually reviewing the code of a suspicious extension was long, because the process was rather imprecise and not straightforward. Then throughout that period of tests, we had acquired a lot of expertise in the review process, which made it faster in the end. For instance, we have started to recognize similar patterns and codes in extensions (many extensions reuse similar code) and therefore we knew when they had to be skipped or not an extension. Currently, we think that for an expert, 15 mn would be a sufficient average time to correctly review an extension for the threats that we have reported. In practice, some extensions will take longer to analyse while others will be analyzed in a couple of minutes.

C. Limitations and Challenges

First note that we considered only scripts that are part of the extension packages. For instance, background and UI pages may reference external scripts. Those scripts were not considered in our analysis. Nonetheless, we think that extensions bundles are more likely to contain most of the APIs that we consider in this work, as extensions developers are recommended to avoid referencing remote scripts in extensions codes.

Our static analysis tool suffers from many limitations. The first one is the fact that we did not consider scopes [37], which lead to unnecessary functions being analyzed. However, this is not a problem ultimately because all the results were further manually reviewed to remove false positives. It also suffers from some false negatives, mainly because of the flexibility of JavaScript that make it challenging to exhaustively address all the ways message listeners can be invoked in extensions.

Manually analyzing complex, large and sometimes minified and obfuscated JavaScript programs making use of callbacks everywhere is not trivial for a single person. But we have taken all the time that was necessary to correctly perform the study. Finally, for a very few extensions, despite all our efforts at the static and manual analysis levels, we could not draw any conclusion about the potential threats they may pose.

IV. Tool for analyzing communications APIs

This section demonstrates a case study of the tool. We have released online a web version of the tool. It can be used to analyzed extensions directly in a browser.

Result of the static analyzer Listing 2 shows the result produced by the tool when applied to the *eRail.in* Chrome extension [41].

```
{
    "com_via_cs": {
        "to_back": {
            "back": {
                "ajax": {
                    "$.get": "",
```

[1]CRX Extension Viewer [39] is an handy extension that can download and nicely display in the browser the content of extensions, allowing to navigate their files very conveniently

[2]Indeed, content scripts are declared as an array of a set of files, so the tool reports the index of the files to analyse

```
                "$.post": "",
                "$.ajax": "",
                "XMLHttpRequest": ""
            },
            "cookies": {
                "chrome.cookies.getAll": "",
                "chrome.cookies.remove": "",
                "cookies": ""
            }
        }
    }
}
```


Listing 2: Result of analyzing the *eRail.in* extension

- `com_via_cs` implies that webpages can communicate with the extension via the content scripts, by using the `postMessage` API. This extension has only 1 content script. When there are multiple content scripts, the tool analyzes each of them independently and produces results corresponding to each of them.
- `to_back` indicates that the messages sent by webpages to the content script are forwarded to the extension background page.
- The tool found that two sensitive APIs are reached in the background page: AJAX requests with calls to the jQuery AJAX APIs (`$.get`, `$.post`, `$.ajax`) and access to cookies with invocation to the `chrome.cookies.getAll` and `chrome.cookies.remove` APIs.

The main goal of the tool is to raise awareness about the fact that an attacker may potentially get access to the extension's privileged APIs. One can then further review the code to validate or refute the results of the tool. For instance, after manually vetting the code of the *eRail.in* extension, we effectively confirm that any webpage can access all user cookies and make AJAX request to any domain. See Section VI for more details about examples of messages to be sent to extensions to benefit from their privileged capabilities.

Releasing the tool In addition to the full static analyzer used in this work, we have also prepared an online version for analyzing the message passing APIs of extensions. The only difference with the version used in this work is that it does not handle dynamically injected content scripts. This was done for simplicity reasons. That notwithstanding, in order to analyze dynamic content scripts, one can simply declare them in the extension manifest as static content scripts. Both versions of the tool will be publicly released.

There is room for further improving the tool. For instance, lessons can be learnt from the state-of-the-art on JavaScript static analysis tools in order to improve the extraction of messages passing listeners and tracking the escalation of extensions sensitive APIs. The set of threats considered in this work can also be extended further with state-of-the-art extensions threats in the literature.

V. Empirical Study

We downloaded Chrome [42], Opera [43], and Firefox [44] extensions by the end of November 2017. The extensions were statically analyzed in the beginning of February 2018 — on a cluster of 200 nodes mainly because of storage limitations on our own devices. This was preceded by a long period of tests during which we improved the static analyzer, and fixed the list of security and privacy threats. In the middle of May 2018, we did another crawl and analysis. The results presented here are for this second dataset.

In this section, we first give an overview of the results, then we discuss in more details each threat and the report extensions where it was found.

A. Overview

Table I presents the number of extensions we collected and analyzed. Chrome provides the largest share of extensions, followed by Firefox and Opera. Recall that for Firefox, we are considering only extensions built using the new WebExtensions API [2], and not those using the XPCOM/XUL API [45].

The static analysis tool reported 3,996 suspicious extensions that we manually vetted. The results of the manual analysis are also shown in Table I. As with the share of extensions, Chrome had the largest share of extensions with threats. In a total of 197 extensions, only 16 were found on Firefox, 10 on Opera, and the 171 others are Chrome extensions. Note that some single extensions pose more than one threat at a time. All the 197 extensions reported here effectively posed at least one or more of the security and privacy threats described in Section II. During the manual analysis, we also identified the messages to be sent in order to exploit their capabilities. The full list of the extensions and the threats that they pose are given in Table IX in the Appendix, because of page limitations.

TABLE I: Data overview

	Chrome	Firefox	Opera	Total
Extensions analyzed	66,401	9,391	2,523	78,315
Suspicious extensions	3,303	483	210	3,996
Execute Code	15	2	2	19
Bypass SOP	48	9	6	63
Read Cookies	8	-	-	8
Read History	40	-	-	40
Read Bookmarks	37	1	-	38
Get Extensions Installed	33	-	-	33
Store/Retrieve Data	85	2	3	90
Trigger Downloads	29	5	2	36
Total of unique extensions	**171**	**16**	**10**	**197**

Extensions installs and categories Figure 3 presents the distribution of users impacted, or the number of installs per extension at the time of writing this paper. Around 55% of the extensions have less than 1000 users, while the remainder 45% have thousands of installs, showing that those threats are present in rather popular extensions, hence affecting many users. About 27% of extensions have less than 100 users and another 27% have between 100 and 1000 users. We see this as an opportunity for a tool such as ours to help improve extensions security, as it can serve to detect potentially malicious extensions while they are not yet very popular among users, thereby limiting their impact on users.

Table II further presents the category of these extensions. Note that the categorization of extensions is not done the same

way by Chrome, Firefox and Opera browsers. Some categories exist only on specific browser, and not on others. Moreover, we found similar (or the exact same) extensions being differently classified depending on the browser. We tried to merge the different categories whenever possible.

As one can observe, `Productivity` is the most popular category among the reported extensions. It is also the most popular category among all Chrome and Opera extensions we have downloaded, and also the most popular category in various datasets in recent studies [23], [22], [18]. This category does not exist on Firefox.

We were surprised by the results that only 15 extensions (7.61%) are classified as `Developer Tools`. Considering the severity of the threats, we were expecting that most of them would be extensions provided for developers to perform some controlled experiments. Since our results represent only a lower bound of the number of extensions potentially posing these risks, it would not be surprising that even more extensions also exhibit similar threats.

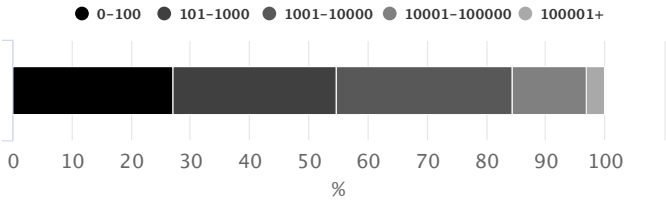

Fig. 3: Distribution of the number of users per extension

TABLE II: Category of extensions

Category	# Extensions
Productivity	81
Social & Communication	48
Fun	19
Accessibility	17
Developer Tools	15
Search Tools	6
Shopping	4
Blogging	2
Privacy & Security	2
Other	2
Appearance	1
Total	**197**

Extensions privilege only some web applications About 55 extensions (45, 7 and 3 on Chrome, Firefox and Opera respectively) communicate with any web applications to give them access to extensions privileged APIs. Interestingly, on Chrome, 7 of them allow to execute arbitrary code in the extension context, 15 of them are concerned with SOP bypass, 26 for storing data, 2 can be exploited by any web application to read all user cookies and 5 to read the cookies of the current web application.

The vast remainder of extensions (72.08%) can be exploited only by specific web apps to benefit from their privileged capabilities. For instance, reading user browsing history, bookmarks, and list of installed extensions, is enabled by extensions only to specific applications such as fliptab.io, atavi.com, mail.google.com. In particular, downloads are allowed by many extensions (on Chrome and Opera) mostly from vk.com.

The fact that most extensions allow communications with only some specific apps can also be explained by the fact that most of those we found allow interactions between web apps and the background pages directly. Let us recall that it is only possible to allow communications between background pages and specific web apps (and not all web apps).

Extensions allow to connect to arbitrary web applications If many extensions tend to privilege specific web applications as shown previously, the exact opposite is observed regarding the hosts extensions allow web applications to connect to, in order to access user data. For example, 37 out of the 48 extensions that can be used to bypass SOP on Chrome, give access to the user data on any other application. On Firefox, it is 6 out of the 9 extensions which allow access to any web application data.

These two observations (extensions mostly give access to their privileged APIs only to some web applications, and allow them to access any other web application data in the case of SOP bypass) suggest that the access they give to their capabilities is rather deliberate. Moreover, for the majority of extensions, the messages to send to exploit the different APIs in extensions are so trivial that they could have only been deliberate (See Section VI).

Most privileged web applications As already mentioned, most extensions allow specific apps to benefit from their privileged APIs. This is the case for instance of fliptab.io where scripts can communicate with 31 very similar HD wallpaper extensions on Chrome, that has hundreds to thousands of users. The domain vk.com can interact with 19 extensions (17 on Chrome and 2 on Opera), mostly to download files on the user device. The domain atavi.com can get access to user's history, most visited websites (topsites) and bookmarks thanks to 6 extensions.

Extensions which pose more than one threat All the extensions reported here pose at least 1 of the security and privacy threats considered in this work. Nonetheless, some extensions pose several threats.

The *eRail.in* [41] extension on Chrome gives access to all user cookies and allows full SOP bypass from any web application. Moreover, it has more than 400k users. Interestingly, a version of the extension exists on Firefox, but it leaks cookies and data of a limited set of web applications (all related to the extension owner's domain) to the the extension's provider own domains. Five extensions provided by Fabasoft (See Table VI in the Appendix) leak the current tab cookies. As such, they allow attackers to even access HTTPOnly cookies, and use them to mount session hijacking attacks.

Ringostat dialer [46] is the only extension that executes arbitrary code sent from app.ringostat.com directly in its background page. All other extensions execute the arbitrary attacker code in the context of the content scripts. Recall that the background page has access to all the capabilities an extension declares. Interestingly, it has the `host`, `storage`, `cookies`, and `tabs` permissions, meaning that any script

present on app.ringostat.com can access user data on any other domain, access the extension storage, cookies, open new tabs, inject code directly in any tab, etc.

StartHQ [47] also allows to bypass SOP from starthq.com, and leaks user browsing history. Similarly, *SalesforceIQ CRM* [48] allows to bypass SOP and leaks installed extensions to mail.google.com and salesforceiq.com.

Finally, user browsing history, bookmarks and installed extensions can be read by an attacker in atavi.com and *.fliptab.io thanks to 6 and 31 extensions respectively (See the full list in the Appendix). The latter also let fliptab.io stores and retrieves data in the extension storage.

Cross-browser extensions It is worth mentioning that most of the extensions we found on Opera and Firefox were also present on Chrome. While the compatibility of extensions APIs on major browsers [2], [1], [3], [4] let developers reach more users, attackers also widen their attack surface because they can impact more users thanks a single cross-browser extension. For instance, we have noticed that *megatest2016*, an extension provider, had 2 extensions on Chrome, and a very similar one on Opera. At the time of writing this paper, Chrome removed the 2 extensions (they were allowing ok.ru and other applications to bypass SOP, but we do not know if their removal were due to the SOP bypass) while on Opera, it is still available as *MegaTest* - [49]. The 2 *Photo Zoom for Facebook* and *Facebook Photo Zoom* Firefox add-ons have similar versions on Chrome, but these do not allow SOP bypass. Similarly, the *ModernDeck* extension is present both on Opera [50] and Chrome [51]). On Opera, it allows to store/retrieve data, while on Chrome it does not. This represent yet another problem of cross-browser extensions. While users of the same extension suffer from security and privacy threats on one browser, on the other browser where the extension is removed or fixed, users do not. Browser vendors, and more importantly users would gain from security and privacy perspectives, if browser vendors share their reviews of extensions with one another, in order to help take similar actions like removing extensions, or updating them to remove threats they pose.

B. Execute Code

Extensions execute in browsers with elevated privileges. From an attacker's perspective, being able to execute arbitrary code in an extension context also gives the attacker access to the extension capabilities. We found 15 extensions on Chrome, 2 on Firefox and 2 on Opera that can be exploited by web apps to execute code in their privileged context. Only one extension on Chrome *Ringostat dialer* [46] executes in its background page, code that it receives from app.ringostat.com. Then it gives access to user data on any application, user cookies, allows code injection in in any tab the user opens, the use of the extension storage, etc. All other extensions execute the attacker's code in the contexts of the content scripts. Even though content scripts have limited access to extensions capabilities, they are not subject to SOP, can store/retrieve data, and more importantly, they have access to the full DOM on the web applications pages in which they are injected.

The extension *iwassa*, present on Opera [52] and Chrome [53] allows any app to open any URL in a new tab, and execute any code (content script) in it. If the code in the context of the content script can already access any application data, one can further inject specific content in the DOM of the new tabs opened, to exfiltrate for instance any token/secret present in the application DOM. In fact, in addition to cookies, many sensitive applications use tokens to further perform additional checks about the origins of requests before letting users perform sensitive actions on their data.

Another interesting example is that of the *LinkClicker* extension also present on Opera [54] and Chrome [55]. It allows any application to send a code which will be further injected in any new tab the user opens during the current browsing session. One can use it to track the user while she is browsing, gather any credentials that she is providing to log into any application, and exfiltrate those to the attacker.

In many of these cases, the problem is due to the fact that the extension does not correctly sanitize the codes received from web applications, allowing attackers to execute arbitrary codes. A good example is that of the *GureTV: To watch television* extension on Firefox [56]. It did well to sanitize content sent from web applications, but not content sent from iframes embedded in the applications. Hence, one can create an iframe, and send an arbitrary code which will be executed in the context of the content scripts.

Many of the other extensions work similarly, and allow (at least) to access arbitrary user data on any application, and/or store and retrieve data (when they have the appropriate permissions).

C. Bypass SOP

Extensions are not subject to the SOP, and therefore have access to user data on any web application for which they have declared the `host` permission. Through message exchanges with extensions, 48, 9 and 6 of extensions on Chrome, Firefox and Opera respectively, allow web applications to bypass SOP by accessing user data on any other web application. As for other threats, the trend is rather to allow only some web applications to bypass SOP, even though 15 of such Chrome extensions allow any application to access any other application data. Hence, the majority of arbitrary SOP bypass can be exploited by specific web applications, including: ok.ru, mail.google.com, logincat.com, etc. Interestingly, when SOP bypass is possible, in most of the cases the data of all domains can be accessed. On Chrome for instance, it is 37 out of the 48 extensions that allow access to any application data. Even when the SOP bypass is partial, it is enabled to rather sensitive domains. For instance, 5 extensions out of 11 allow SOP bypass to users' Google accounts: salesmate.io, appspot.com and aliexpress.com can access users Gmail account. One extension [57] allows access to the linkedin.com data of more than 400k users from Gmail, and blog.renren.com can access github.com [58].

D. Cookies

We found 8 Chrome extensions that can be exploited by web applications to read user cookies: 2 of them allow any web application to read all user cookies [41], [59], 1 only allow `app.ringostat.com` [46] to read all user cookies, and the other 5 of them allow an attacker script to read the cookies of the tab in which it executes. The number of users affected is very important (more than 415k for *eRail.in* [41], 9.6k for *Telerik Test Studio Chrome Playback 2014.1* [59] and 78 for *Ringostat dialer* [46]. Cookies can be used to hijack users browsing sessions, access their data and take actions on their behalf. It is worth mentioning that the three extensions that can be exploited to read all user cookies, have probably been poorly programmed. It is more likely that the ability to read cookies was meant to be used from specific web applications, but unfortunately the extensions were poorly programmed, allowing other web applications to also get access to user cookies. In particular, the *Ringostat dialer* [46] extension did not expose any means to get user cookies. But it allows to execute any code sent from app.ringostat.com in the extension background page context (using `eval` function), giving the application access to all the capabilities of the extension. Among those, the cookies, storage and arbitrary host permissions, and the ability to open tabs, inject and execute arbitrary code in them, etc.

We found that the web application https://erail.in/ is effectively reading all user cookies when the *eRail.in* [41] Chrome extension is installed. This means that the extension is intentionally given access to user cookies to https://erail.in. However, it is not clear whether the cookies of interest were only those of https://erail.in or any cookie or if only cookies of https://erail.in/ were meant to be leaked. Unfortunately, any web app can access all user cookies stored by any web application, and use them to hijack user sessions. Interestingly, the extension has a version on Firefox, where the cookies which are leaked are only those of domains related to erail.in and are leaked only to erail.in and eair.in.

The case of the extension *Telerik Test Studio Chrome Playback 2014.1* [59] is particularly interesting, as one has to setup complex interactions, involving the extension content scripts and background page, as well as the application and its server. In particular, the interactions are triggered from the web application, but the cookies are sent to the server of the application instead of being returned directly to the application. Following the same mechanism, one can clear cookies, delete user browsing history, etc. A similar extension is also available on Firefox *progress-test-studio-extension*. Unfortunately, we could not analyze it as it was not downloading.

Finally, 5 Fabasoft extensions (See more details in Table VI in the Appendix) allow the attacker to read the current tab cookies of any web application. Even though a web application protects its cookies with the HTTPOnly flag [60], an attacker script running in the web application bypasses this protection by obtaining the cookies via the extension. It can further use them to mount session hijacking attacks against the user.

E. Downloads

Exploiting extensions to trigger the download of arbitrary files is enabled mainly from specific applications including vk.com (See Table IV in Appendix) and ok.ru. Only 2 extensions on Chrome and 3 on Firefox allow downloads from arbitrary web apps. The main purpose of the related extensions were to allow the download of music and videos. Sometimes, they would even suffix the downloaded file name by `.mp3` or `.mp4`. Nonetheless, we have been able to exploit these extensions in order to trigger the download of arbitrary files and save them in the user's device. An attacker can also do so to download malicious software, which when inadvertently executed by the user, may allow the attacker to take control of their computer and perform malicious actions.

It is worth mentioning that none of these extensions required user action to trigger the downloads. One of them, *multiDownloader* [61] even overwrites a file if it is already present on the user's device.

It is also worth mentioning the case of the Chrome *repl.it download* extension [62]. It is a helper extension for the https://repl.it application used for creating and running programs in different languages online. The extension allows to save the code being created. Even though the extension prompts the user to confirm the file name (default is `program.`), the content of the file can be fully arbitrary. As such, an attacker can trick the user in saving the code being edited, while a completely different content is saved.

F. History, Bookmarks, and List of Installed Extensions

Two providers distinguish themselves with regards to extensions that can be exploited to get access to user browsing history, bookmarks and list of extensions. On Chrome, `fliptab.io` [63] provides 31 very similar HD wallpapers extensions (See the full list in Appendix), and allows fliptab.io to get all browsing history, bookmarks and the list of user installed extensions. Each of these extensions has between a hundred and 25k users.

Furthermore, six extensions provided by atavi.com also provide the same privileges to pages at atavi.com and atavi.test. One of them, *Atavi - bookmark manager* [64] has more than 96k users.

Additionally, *Browser History* [65] leaks user browsing history to www.americaninternetmatrix.com/history. Finally, *StartHQ* [47] leaks browsing history to https://starthq.com. Other extensions that give access to the list of extensions include *Boomerang for Gmail* [66] (with more than 1.5 million users) to mail.google.com and *SalesforceIQ CRM* [48], to mail.google.com and salesforceiq.com.

G. Store/Retrieve Data

About 85 extensions can be exploited by various web applications to store and retrieve data. On Chrome, 26 of these extensions give any application access to their storage. Others give specific apps access to their storage. For instance, `fliptab.io` can store data in the user's browser thanks to its 31 extensions. The domain `netflix.com` is also able to

store data thanks to 3 extensions, and `mail.google.com` to do so thanks to 2 extensions. The extensions *ISOGG Y-Tree AddOn* [67] and *PhyloTreeMT AddOn* [68] are from the same provider, even though the web applications they allow to persist data are respectively `isogg.org` and `phylotree.org`.

Recall that extensions storage is persistent and not affected by the clearing of browsing data (web application cookies, storages, ...). As such, they represent a resilient storage which can be used to bypass user privacy preferences and uniquely identify them even though they have cleared their cookies. Interestingly, some extensions propose to sync data they store on all the devices the user is logged into. For instance, if a user logs into multiple devices with the same extension installed, then syncing storages lets an application tracks her accross all her devices.

H. Other Threats

For SOP bypass, we have reported here the cases where web applications can access arbitrary data on other web applications. Nonetheless, we found many extensions allowing to access some predefined data of other web applications. This also represents a SOP bypass (since web applications cannot access such data with their normal privileges). Finally, we found some Opera and Chrome extensions (like the 31 HD wallpaper extensions by fliptab.io), and some not reported here) which allow web applications to clear user browsing data including cookies (or even set/get cookies of some specific domains), history, bookmarks, cache, stored passwords, or enable/disable/uninstall extensions. We do not include such cases in this paper because of page limitations.

VI. Case Study

In this section, we show how an attacker can exploit the capabilities of an extension by sending the appropriate message. One can also find online a few videos demonstrating the exploits on some of concerned extensions, on the Chrome browser. In order to gain access to privileged browser features via an extension, an attacker first needs to ensure that the extension is installed and enabled. Many recent studies discussed extensions discovery, using for instance their unique identifiers and web accessible resources [21], [23] or DOM specific changes they introduce in web pages [22]. This is not really needed here. Knowing the structure of messages extensions respond to, is sufficient. If the extension is present, it will surely reply. To benefit from extensions capabilities, it is sufficient that the attacker is present in a web application with which the extension can interact.

A. Example of messages to send to extensions

We refer to Section II which presents the message passing APIs between webpages and the different components of an extension. Because of page limitations, we cannot provide for all extensions, the messages that can be sent from web pages to exploit extensions capabilities. We illustrate at least each threat by an extension.

Execute code in content scripts context: Listing 3 present the structure of messages that can be sent from any webpage to the *jianlibao* [69] Chrome extension to execute arbitrary code in the context of its content scripts. Replace `CODE` with real JavaScript code, then serialize the message using `JSON.stringify` before sending it. The extension has the `storage` and `host` permissions meaning that any page can bypass SOP and get access to user data on any domain, store data in the extension storage and later retrieve it for tracking purposes. Moreover, the code is injected in the active tab the user is interacting with. As the user may switch tabs at any time, one can send the code regularly (say every second) in order to ensure that it is injected in all the web applications the user is interacting with. Since content scripts have access to the DOM of webpages, the injected code also has full access to the active tab DOM, giving it the ability to undertake any action: recording user name and password, credit card numbers, emails, etc.

```
{
  type: "getResumeInfo",
  downloadObj: {
        resumeWhereabouts: 5
    },
    context: {
        contentScript: CODE,
        jsMethod: "console.log"
    }
}
```

Listing 3: Executing arbitrary code in the context of the content scripts of the current tab the user navigates to, thanks to the *jianlibao* Chrome extension.

Extensions such as *iwassa* [53], [52] or *LinkClicker* [55], [54], present on Chrome and Opera, even allow to send a URL and a code. They will open the URL in a new tab, and execute the code in the context of the content scripts injected by the extension in the new tab. Listing 4 presents the case of the *iwassa* extension. Replace *URL* with the URL of the page to open in a new tab, and `CODE` with the real code to be executed in the context of the new tab content scripts.

```
{
  from: "logininfo",
  val: [URL, CODE, "LoginAPI"]
}
```

Listing 4: Executing code in the context of a choosen tab thanks to the *iwassa* extension present on Chrome and Opera. *URL* is the URL of the page to open in a new tab, and `CODE` the code to be executed.

The extension also has the `host` permission, allowing to make AJAX requests to any domain.

Execute code in background page context: Background pages are the most privileged contexts, as they have access to all the capabilities of an extension. Listing 5 shows the message to send to the *Ringostat dialer* [46] Chrome extension to execute arbitrary code in the context of its background page. Interestingly, this extension has the `host`, `storage`, `cookies` and `tabs` permission, giving an attacker the ability to bypass SOP, store data in the extension storage, manage

user cookies and tabs (open new tabs, close some, etc.). Messages are to be sent from webpages which URLs match `*://app.ringostat.com/*`.

```
{
  message: "execCommand",
  data : {
    command: "eval",
    params: CODE
  }
}
```

Listing 5: Message to send to *Ringostat dialer* background page to execute arbitrary code. Replace `CODE` with the real code to be executed.

Bypass SOP: Here we take the example of the *Buxenger* extension, available both on Chrome and Firefox. Listing 6 shows the structure of messages to be sent to the extension in order to make AJAX requests to any domain (SOP bypass). The case shown here, is for making HTTP *GET* requests. But the extension also allows to make AJAX requests using HTTP *POST, DELETE, PATCH* methods.

```
{
  message: "ajax-get",
  url: URL,
  callbackId: ID
}
```

Listing 6: Make arbitrary AJAX requests thanks to the *Buxenger* extension present on Chrome and Firefox. Replace *URL* with the URL of the data to access, and *ID* with any value.

Retrieve cookies: Listing 7 shows the case of the *eRail.in* Chrome extension which allows any webpage to retrieve the list of user cookies.

```
{
  Action: "GETCOOKIE"
}
```

Listing 7: Message to send to *erail.in* extension in order retrieve all user cookies

This includes any cookies, such as the user authentication cookies set after she has logged into web applications. One can further use the cookies to mount session hijacking attacks. The extension also allows to make arbitrary AJAX requests, by sending messages as shown in Listing 8

```
{
  Action: "GET_BLOB",
  URL: URL
}
```

Listing 8: Making AJAX requests thanks to the *eRail.in* Chrome extension

Downloads files: Listing 9 shows the signature of messages to send from any webpage, to the *HTTP Commander* [70] Chrome extension in order to trigger the download of any file. Replace *FILE_URL* with the URL of the file to download, and *FILE_NAME* with the name under which the file will be saved on the user device. Multiple files can be sent in the message. They will all be downloaded one after the other.

```
{
  type: "HTCOMNET_DOWNLOAD",
  files: [{
    url: FILE_URL,
    path: FILE_NAME
  }]
}
```

Listing 9: Download files on the user device, thanks to the *HTTP Commander* extension.

Store data in extension storage: Listing 10 shows messages to send in order to store and retrieve data in the *VisualSP Training for Office 365* [71] Chrome extension storage. Replace *DATA_TO_STORE* with the data to be stored in the extension storage. Later on, send the second message to retrieve data. The data will be sent to iframes in the page. To collect the data previously stored in the extension storage, before sending the message, one can simply add an iframe to the webpage, then send the message, collect the previously stored data from the iframe, and send it back to the parent page.

```
// Store data
{
  owner: "VisualSP",
  command: "SetUserId",
  data: DATA_TO_STORE
}
// Retrieve data.
{
  owner: "VisualSP",
  command: "GetUserId"
}
```

Listing 10: Store and retrieve data in *VisualSP Training for Office 365* Chrome extension storage

History, bookmarks, extensions list: We show here the case of the *Space Galaxy HD Wallpapers* [72]. It is one of the 31 HD Wallpapers from `fliptab.io` (See Table III in the appendix) that lets pages matching *.fliptab.io, to manage user history, bookmarks, extensions list and storage. Listing 11 shows the different messages that has to be sent to get the related information.

```
// Message for retrieving user browsing history
{
  type: "history",
  act: "get_all"
}

// Message for retrieving bookmarks
{
  type: "bookmarks",
  act: "get_all"
}

// Message for retrieving the list of extensions
{
  type: "extensions",
  act: "get_all"
}
```

Listing 11: The *Space Galaxy HD Wallpapers* Chrome extension allows to get user browsing history, bookmarks and extension list

B. Forcing the attack

In order for an attacker to gain access to an extension's APIs, he must have a script loaded in a web application that is allowed to interact with the extension. Moreover, in most cases, the application has to be running in the user browser in order for communications to be possible. Figure 4 shows a simple scenario in which A.com is an application currently running in a user browser. This application provides content A.com/content (a script) for another application B.com which can communicate with an extension to get access to some privileged APIs. However, B.com is not currently running in the user browser. A.com can force the attack to happen, by opening B.com (upon a user interaction with the A.com). Once B.com runs, the script that it embeds from A.com gets executed and can communicate with the extension to get access to its privileged APIs — for instance to access user data on any other application — and exfiltrate this to A.com. With the prevalence of some third party scripts providers among web applications [73], this scenario can be easily implemented by attackers to gain from extensions capabilities.

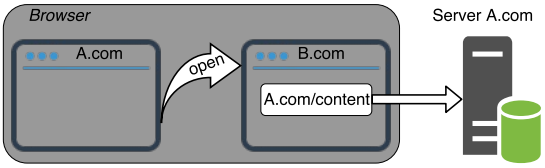

Fig. 4: A.com forces an attack by opening B.com thereby allowing A.com/content to load, execute and interact with extensions in order to exfiltrate user data to A.com.

Combining multiple extensions Another scenario where access to any extension capabilities can be indirectly gained is when some extensions make it possible to open new tabs and inject and execute arbitrary codes in them. We have recorded a video showing the use of the *LinkClicker* extension [55] which allows to open a new tab and execute code in it, and the *Space Galaxy HD Wallpapers* extension [72] which allows only `fliptab.io` to get/delete user browsing history, bookmarks and extensions list. From any application (the `localhost` in our example), we opened `www.flipatab.io`, and injected a code in its context. The code retrieved the list of extensions, bookmarks and user history. This information could be further sent to a server chosen by the attacker. One can even use the *LinkClicker* extension to send the retrieved information back to the attacker by opening a new tab of the attacker application (`localhost` in our case).

VII. Discussion

Here we discuss countermeasures and proposals to mitigate the threats introduced by extensions via message passing.

A. Disclosure to vendors

We have disclosed the list of extensions to Chrome, Firefox and Opera. All vendors acknowledged the issues. Firefox has removed all the reported extensions. Opera has also removed all the extensions but 2 which can be exploited to trigger downloads. The reason given by Opera is that the downloads can only be triggered from specific websites. However, we made them observe that those websites include third party scripts that can also trigger arbitrary downloads. So discussion still continues with Opera on the 2 remaining extensions, in particular to ensure that users are aware of the downloads. Chrome also acknowledged the problem in the reported extensions. We are still discussing with them on potential actions to take: either remove or fix the extensions.

B. Proposals

The discussions with browser vendors confirmed our arguments that their current extensions review process is weak. In fact, none of them has considered the fact that extensions put user data at risk via vulnerabilities in the use of message passing APIs. Moreover, we are worried that malicious extensions developers that would be aware of the ability to exfiltrate user data via message passing, would deliberately introduce such vulnerabilities in their extensions. There are various ways an extension could exploit its own vulnerabilities without being blocked by browser vendors. For example, the extension developer can operate a website. Then when the user opens her browser and navigates to her favorite web applications, the extension injects its own website as an iframe and exfiltrate user data from that iframe. For browser vendors, a quick fix of the threats discussed in this work is to consider message passing interfaces as a medium for introducing vulnerabilities in extensions, thereby putting users data at risk. New extensions must be reviewed accordingly, in order to fix such threats. To help in this process, browser vendors may mandate that extensions explicitly declare the list of web applications they intent to interact with by message passing via the extensions content scripts, very similarly to what is done with the `externally_connectable` key used in extensions `manifest.json` files to declare the list of web applications the extension background pages intent to directly interact with (See Section II-B).

The best solution to mitigate this threat would have been to ban the interactions between webpages and extensions, but this would impact the many extensions making use of these communication interfaces. Nonetheless, the needs and implementations of the message passing interfaces are questionable. In fact, extensions can already read/write web applications DOM. For extensions that absolutely need to exchange messages with webpages, browser vendors may review the current extensions system and allow messages only from code injected by extensions in the context of webpages. In fact, extensions can inject code directly in the context of webpages. Currently, such code runs with the same privileges as codes loaded by webpages themselves. We envision an architecture in which the browser tracks the origin of messages received in extensions. And if they are not sent by code injected by extensions, the messages is not delivered to the extension. There are surely ways an attacker can circumvent this solution,

but such attacker is exactly the one already discussed by Carlini et al. [10] and Bandhakavi et al. [9].

VIII. Related Work

The security and privacy implications of browser extensions have been extensively studied. Barth et al. [8] analyzed the Firefox XPCOM architecture and proposed a new extensions architecture that has since been adopted by Google Chrome and evolved into the Chrome Extensions API compatible with the cross-browser WebExtensions API. Before them, many authors had also shown the dangers of misusing the powerful APIs provided to Firefox XPCOM extensions and propose tools for discovering vulnerabilities and securing extensions [7], [9], [11], [12]. Among other things, the permissions system in extensions was meant to reduce extensions capabilities, and hence reduce the harms that attackers can cause if they compromise an extension. However a good number of studies have shown that many extensions still request too many permissions [14], [15], [16]. Guha et al. [14] proposed IBEX a cross-browser extensions platform, supporting fine-grained access control policies with tools for verifying the compliance with the security policies. The work of Carlini et al. [10] on vulnerable extensions has led to the ban of inline and HTTP scripts and eval-like functions in extensions background pages. Different dynamic analysis systems have been proposed for discovering malicious extension such as Hulk [15] and Ex-Ray [17] based on the concept of honey pages. Starov and Nickiforakis [18] found many extensions leaking sensitive user information such as browsing history, search queries, form data and extensions list. Recently, many studies have demonstrated different techniques for discovering browser extensions and shown that they can be used to fingerprint browsers [21], [22], [23], [24]. Other threats considered in this paper have been discussed outside of browser extensions [74], [75], [76], [77], [20], [78], [20], [5], [60].

Calzavara et al. [13] were the first to show that message passing interfaces could lead to privilege escalation and exploits by web applications. We discussed directly with the authors of this work. Their goal was to formalize the privileges that an opponent can escalate thanks to the message passing APIs between web applications and extensions content scripts. In the extensions system they considered, content scripts had no privileges and direct interactions with background pages were not possible. They had proposed a prototype implementation of their system named CHEN for developers to evaluate the robustness of an extension against privilege escalation and help them refactor their codes, but the tool is no more available and it did not take into account long-term communications (ports)

To the best of our knowledge, this work is the first large-scale study on the security and privacy implications of the communications between browser extensions and web applications, allowing the latter to benefit from extensions privileged capabilities. We built a static analyzer for analyzing extensions and identified a good number of them, demonstrating how these extensions can be exploited by web applications to benefit from extensions privileged capabilities and thereby access sensitive user information.

IX. Conclusion

Browser extensions are third party code in browsers with access to privileged APIs not accessible to web applications. Nevertheless, web applications and browser extensions can interact with one another by exchanging messages. In this paper, we built a static analyzer and applied it to Chrome, Firefox and Opera extensions. We identified a good number of extensions that can be exploited by web applications to benefit from their privileged capabilities. In particular, some vulnerable extensions allow web applications to bypass the Same Origin Policy security mechanism and access user data on any web application. Extensions also leaked user credentials (cookies), browsing history, bookmarks, list of installed extensions, to web applications or allowed them to download any file on the user device, or store data in the extension storage for tracking purposes. We showed how trivially, attackers can exploit those threats, and discussed proposals as to mitigate them. In particular we argued for a review process taking into consideration the threats we have discussed, with the help of tools such as our static analyzer, or changes in the extensions system itself to ban or limit messages only to extension injected scripts.

Acknowledgments

We would like to thank the anonymous reviewers and the PC for the quality of the comments and revision expectations as they helped us improve the work. Special thanks to Nadege Somé, Sadry Fievet, Nataliia Bielova, and Tamara Rezk for proof-reading, support and insightful discussions, comments and suggestions. The author is grateful to Inria Sophia Antipolis - Méditerranée "Nef" computation cluster for providing resources and support. This work was supported by the project PIA ANSWER.

References

[1] Chrome Extensions API. [Online]. Available: https://developer.chrome.com/extensions

[2] Mozilla WebExtensions API. [Online]. Available: https://developer.mozilla.org/en-US/Add-ons/WebExtensions

[3] Opera Extensions API. [Online]. Available: https://dev.opera.com/extensions/

[4] Microsoft Edge Extensions API. [Online]. Available: https://docs.microsoft.com/en-us/microsoft-edge/extensions

[5] Same Origin Policy. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[6] Cross-Origin Resource Sharing. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[7] M. T. Louw, J. S. Lim, and V. N. Venkatakrishnan, "Extensible web browser security," in *Detection of Intrusions and Malware, and Vulnerability Assessment, 4th International Conference, DIMVA 2007, Lucerne, Switzerland, July 12-13, 2007, Proceedings*, ser. Lecture Notes in Computer Science, B. M. Hämmerli and R. Sommer, Eds., vol. 4579. Springer, 2007, pp. 1–19. [Online]. Available: https://doi.org/10.1007/978-3-540-73614-1_1

[8] A. Barth, A. P. Felt, P. Saxena, and A. Boodman, "Protecting browsers from extension vulnerabilities," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/10/pdf/04.pdf

[9] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, "VEX: vetting browser extensions for security vulnerabilities," in *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. USENIX Association, 2010, pp. 339–354. [Online]. Available: http://www.usenix.org/events/sec10/tech/full_papers/Bandhakavi.pdf

[10] N. Carlini, A. P. Felt, and D. A. Wagner, "An evaluation of the google chrome extension security architecture," in *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, T. Kohno, Ed. USENIX Association, 2012, pp. 97–111. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/carlini

[11] K. Onarlioglu, M. Battal, W. K. Robertson, and E. Kirda, "Securing legacy firefox extensions with SENTINEL," in *Detection of Intrusions and Malware, and Vulnerability Assessment - 10th International Conference, DIMVA 2013, Berlin, Germany, July 18-19, 2013. Proceedings*, ser. Lecture Notes in Computer Science, K. Rieck, P. Stewin, and J. Seifert, Eds., vol. 7967. Springer, 2013, pp. 122–138. [Online]. Available: https://doi.org/10.1007/978-3-642-39235-1_7

[12] K. Onarlioglu, A. S. Buyukkayhan, W. K. Robertson, and E. Kirda, "SENTINEL: securing legacy firefox extensions," *Computers & Security*, vol. 49, pp. 147–161, 2015. [Online]. Available: https://doi.org/10.1016/j.cose.2014.12.002

[13] S. Calzavara, M. Bugliesi, S. Crafa, and E. Steffinlongo, "Fine-grained detection of privilege escalation attacks on browser extensions," in *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, 2015, pp. 510–534. [Online]. Available: https://doi.org/10.1007/978-3-662-46669-8_21

[14] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy, "Verified security for browser extensions," in *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*. IEEE Computer Society, 2011, pp. 115–130. [Online]. Available: https://doi.org/10.1109/SP.2011.36

[15] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, "Hulk: Eliciting malicious behavior in browser extensions," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 641–654. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kapravelos

[16] S. Heule, D. Rifkin, A. Russo, and D. Stefan, "The most dangerous code in the browser," in *15th Workshop on Hot Topics in Operating Systems, HotOS XV, Kartause Ittingen, Switzerland, May 18-20, 2015*, G. Candea, Ed. USENIX Association, 2015. [Online]. Available: https://www.usenix.org/conference/hotos15/workshop-program/presentation/heule

[17] M. Weissbacher, E. Mariconti, G. Suarez-Tangil, G. Stringhini, W. K. Robertson, and E. Kirda, "Ex-ray: Detection of history-leaking browser extensions," in *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*. ACM, 2017, pp. 590–602. [Online]. Available: http://doi.acm.org/10.1145/3134600.3134632

[18] O. Starov and N. Nikiforakis, "Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions," in *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich, Eds. ACM, 2017, pp. 1481–1490. [Online]. Available: http://doi.acm.org/10.1145/3038912.3052596

[19] Cross-Origin Communications. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

[20] Session Hijacking Attack. [Online]. Available: https://www.owasp.org/index.php/Session_hijacking_attack

[21] A. Sjösten, S. V. Acker, and A. Sabelfeld, "Discovering browser extensions via web accessible resources," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017*, G. Ahn, A. Pretschner, and G. Ghinita, Eds. ACM, 2017, pp. 329–336. [Online]. Available: http://doi.acm.org/10.1145/3029806.3029820

[22] O. Starov and N. Nikiforakis, "XHOUND: quantifying the fingerprintability of browser extensions," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 941–956. [Online]. Available: https://doi.org/10.1109/SP.2017.18

[23] I. Sánchez-Rola, I. Santos, and D. Balzarotti, "Extension breakdown: Security analysis of browsers extension resources control policies," in *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, E. Kirda and T. Ristenpart, Eds. USENIX Association, 2017, pp. 679–694. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/sanchez-rola

[24] G. G. Gulyás, D. F. Somé, N. Bielova, and C. Castellucia, "To extend or not to extend: on the uniqueness of browser extensions and web logins," in *To appear in the Proceedings of the 2018 ACM on Workshop on Privacy in the Electronic Society, WPES@CCS 2018, Toronto, Canada, October 15 - 19, 2018*, 2018.

[25] Document Object Model (DOM). https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.

[26] Extensions and the add-on ID. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/WebExtensions_and_the_Add-on_ID

[27] Browsing Contexts. https://www.w3.org/TR/html51/browsers.html.

[28] Chrome Extensions - Message Passing. [Online]. Available: https://developer.chrome.com/extensions/messaging

[29] Opera - Passing Messages in Extensions. [Online]. Available: https://dev.opera.com/extensions/message-passing/

[30] A. Hidayat. ECMAScript Parsing Infrastructure. [Online]. Available: https://www.npmjs.com/package/esprima

[31] B. Newman. JavaScript Syntax Tree Transformer. [Online]. Available: https://www.npmjs.com/package/recast

[32] HTML Parser Implemented in JavaScript. [Online]. Available: https://www.npmjs.com/package/jsdom

[33] SlimerJS - A scriptable browser for Web developers. [Online]. Available: https://slimerjs.org/

[34] Abstract Syntax Tree. [Online]. Available: http://esprima.readthedocs.io/en/4.0/syntax-tree-format.html#expressions-and-patterns

[35] ECMAScript 2017 Internationalization API Specification (ECMA-402, 4th Edition, June 2017). https://www.ecma-international.org/ecma-402/4.0/.

[36] Window . [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window

[37] JavaScript scope. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Scope

[38] JavaScript Object Property Access - Dot and Array Notation. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_Accessors

[39] R. Wu. CRX Extension Source Viewer For Chrome, Opera, and Firefox. [Online]. Available: https://robwu.nl/crxviewer/

[40] Browser Console. https://developer.mozilla.org/en-US/docs/Tools/Browser_Console.

[41] eEail.in Chrome extension. [Online]. Available: https://chrome.google.com/webstore/detail/erailin/aopfgjfeiimeioiajeknfidlljpoebgc

[42] Chrome Extensions. [Online]. Available: https://chrome.google.com/webstore/category/extensions?hl=en-US

[43] Opera Add-ons. [Online]. Available: https://addons.opera.com/en/extensions/

[44] Firefox Add-ons. [Online]. Available: https://addons.mozilla.org/en-US/firefox/

[45] XPCOM Interfaces. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/XPCOM_Interfaces

[46] Ringostat dialer. [Online]. Available: https://chrome.google.com/webstore/detail/ringostat-dialer/pfofjhnkanlacmgfgjohncmgemffkldl

[47] StartHQ. [Online]. Available: https://chrome.google.com/webstore/detail/starthq/ilcpdgfepihaomggobhmfiimflngbcoh

[48] SalesforceIQ CRM. [Online]. Available: https://chrome.google.com/webstore/detail/salesforceiq-crm/jpcebpeheognnbogfkpllmmdnimjffdb

[49] MegaTest - - Opera Extension. [Online]. Available: https://addons.opera.com/en/extensions/details/megatest-uznat-rezultat/

[50] ModernDeck - Opera Extension. [Online]. Available: https://addons.opera.com/en/search/?query=lkdpdiepahdagdknbbjgnadholcdgfib

[51] ModernDeck - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/moderndeck/pbpfgdgddpnbjcbpofmdanfbbigocklj

[52] IWASSA - Opera Extension. [Online]. Available: https://addons.opera.com/en/search/?query=bmjcngclkmgpfbjcmnbidognkoocpllm

[53] Iwassa - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/iwassa/hnkmipajjgbclkombnmigfnpekddlhlh

[54] LinkClicker - Opera Extension. [Online]. Available: https://addons.opera.com/en/search/?query=jnmcfakfglphcmgokeeoihifcenjjcgg

[55] LinkClicker - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/linkclicker/hoobpdocllliidciecjifpikpnopjpmkh

[56] GureTV: To watch television - Firefox Extension. [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/guretv-ver-tv/

[57] LinkedIn Sales Navigator - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/linkedin-sales-navigator/hihakjfhbmlmjdnnhegiciffjplmdhin

[58] renren-markdown - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/renren-markdown/iiabjaofopjooifoclbpdmffjlgbplod

[59] Telerik Test Studio Chrome Playback 2014.1. [Online]. Available: https://chrome.google.com/webstore/detail/telerik-test-studio-chrom/pkkbbimilpjmghfhhppamgigileopnkc

[60] Http cookies. [Online]. Available: https://developer.mozilla.org/fr/docs/HTTP/Cookies

[61] multiDownloader - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/multidownloader/dnohbnpecjinmdpeikpnmheeepnapfci

[62] repl.it download - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/replit-download/pgmcojeijjhacgkkjaakdafmloncpema

[63] HD Wallpapers from fliptab.io. [Online]. Available: http://www.fliptab.io/

[64] Atavi - bookmark manager. [Online]. Available: https://chrome.google.com/webstore/detail/atavi-bookmark-manager/jpchabeoojaflbaajmjhfcfiknckabpo

[65] Browser History. [Online]. Available: https://chrome.google.com/webstore/detail/browser-history/bpkphnbpiagbpinglgejckickdgaghjo

[66] Boomerang for Gmail - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/boomerang-for-gmail/mdanidgdpmkimeiiojknlnekblgmpdll

[67] ISOGG Y-Tree AddOn - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/isogg-y-tree-addon/cfnjeahambijfdljfacldifapdcklhnj

[68] PhyloTreeMT AddOn - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/phylotreemt-addon/ilpkhojfiejdbkgcjbmllngjebdoehim

[69] jianlibao - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/jianlibao/fimckmjeammfdcpldmcigeojkkmeeian

[70] HTTP Commander - Chrome Extension.

[71] VisualSP Training for Office 365 - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/visualsp-training-for-off/ohdihpdgfenligmhnmldmiabdhflokkh

[72] Space Galaxy HD Wallpapers - Chrome Extension. [Online]. Available: https://chrome.google.com/webstore/detail/space-galaxy-hd-wallpaper/dkpndikhfepllbpaafgcelembimabofo

[73] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. V. Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "You are what you include: large-scale evaluation of remote javascript inclusions," in *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, T. Yu, G. Danezis, and V. D. Gligor, Eds. ACM, 2012, pp. 736–747. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382274

[74] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, S. D. Gribble and D. Katabi, Eds. USENIX Association, 2012, pp. 155–168. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/roesner

[75] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: Policy and technology," in *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pp. 413–427. [Online]. Available: http://dx.doi.org/10.1109/SP.2012.47

[76] S. Englehardt and A. Narayanan, "Online tracking: A 1-million-site measurement and analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1388–1401. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978313

[77] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner, "Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, T. Holz and S. Savage, Eds. USENIX Association, 2016. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lerner

[78] M. Johns, "Session hijacking attacks," in *Encyclopedia of Cryptography and Security, 2nd Ed.*, H. C. A. van Tilborg and S. Jajodia, Eds. Springer, 2011, pp. 1189–1190. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_661

TABLE III: Extensions with the same code base which gives *.fliptab.io access to browsing history (get/delete), bookmarks (get), extensions (get/enable/disable/uninstall) and storage

bddmmehmgpjhhmbbmngdjhlednmkbken, cajmbfbhhfelhgolhldhhodkclpakcfe
cepmfckfppjpbkjgnpokojedlngflnca, clkodoejadlbjaopcjoijihebbgipjff
dekpebffaadijeaogggfhjemdbjgbcao, dkpndikhfepllbpaafgcelembimabofo
eeiedbnahjonkmimigblgchlefcklhok, efdddbobcofamdjmekphjlhgmcnhobbp
ehmhopjniedignnkdeijmpmodhcppgif, eilbnnflfpkhhfmhmlhflhecceajpkcj
fieoemdbopiialnojhifcndkenhjkbmm, fkpmpnljocdllgmplhnmjhjmmilbnofj
gfgchcclfmppnfoakdlhgdhnolbpiedf, glfbbjdfmmlanpikdedpjoeimlijjcjj
hmbedbiicehadpbhbipafffieolpjolh, hocncjdhccalpmblkpagbmjebkfkibbm
iamlligjelallbdddajmbojjjhadkmcf, jcffnpjkbahanenhcnhhdfopkjlpflfm
jokpapkhjeahjbkemfjfhjgcogmbcpoi, kkejopfphkmldfpdmcljfoinfcljijjf
klfeojnepdoehgddffbcjiamcjjahmgj, lbfidebeingoondbmpeapjoeeoloanak
lgphbplfjpemcghfcoajehcmikflcbbd, lmbcpiodajlbgmjbiajgcjdalgbofcbn
loggojfoonblkkhkjpijapeheoogagki, lpkfidfkgflpbakdnhpojiejlpdanknh
mgmodhbknbfmpjmilankiffnjbelcipo, mibaeahdcconphmdndbeipegldkkbcjh
odpiaedkmdpcheddbkilnkelhhocoenn, pfdaccgdljiifplhfnjcacapfednngonb
afddmpnodjaifgjibafjcbfaplnoipei

TABLE IV: Extensions with the same code base for triggering downloads from vk.com, *.vimeo.com, *.coub.com, *.kinopoisk.ru

nfhipbkhabgmkhahoaagkcgppcjikjgl idenapkfefkbknhbmfgeaclpcpbhcnbe
fnnlocjimhjpmgfjhjamdkjhemfhkhjo lmlnplkfbiihcpkghkkmfefjdaccmbcc
kbiocjbkoohjjkkeaafiemjeidgalllh dccmnjciogmmahaogjgkocongokmieog
ekfkljjojhnnhfedepfnbhhfjklagngk hhfgpbjpilbbaomjmdpnfchbpipehiif
pgajmafmbajahclonccaoaoleghhnpam ipeeopcjpgcbgnfogjlickeilmkbonen
jfpmehlefcchhhmlmennihbbihaolabk kcollknpphnodcjdkcmgpjmlbaenabao
backekeabechifnekobfachchocbmjag mfpbgndgoogfplejodpbhnfmaibnalkf
ojhheobonaamlhlcdngacakdcigpeokl mienmjdbnnpaigifneeiifdbjkdgelha
amaobfendgcolppeioeageanmillkmkc

TABLE V: Extensions with the same code base which leaks topsites, history and/or bookmarks to *.atavi.com, *.atavi.test

iglbnbabjdfaobglhonmnlkdbommiebd, knflcnelciofoghldagpknelepafjeif
lamnafpjcnoclihgpefhdbefcmjikhaj, jffjjdoccjiflmckicphblggbppfgklk
ofmacdiceehcibkfednmgpkhgfhpacgi, jpchabeoojaflbaajmjhfcfiknckabpo

TABLE VI: Extensions with the same code base, provided by Fabasoft, which give access to the current tab cookies

ajlbdflhaaflcepndpkdgejimggjcpnm, ngbcdblbfdpjgpmgfagkfofcjbnggfgn
pdhjoolhbkmlgjfedckdhiknnoabbnkk, hiejidhjgjpelfgldfhmnaoahnephhfg
icjlkccflchmagmkfidekficomdnlcig

TABLE VII: Extensions which give access to their storage to any application

eljhpoopiapggnlfcilpbihgbgbpnkgd, akhamklknibionleflabebgeikdookmp
hebabhddakflgmlhgefakkfkciijliie, ilgdjidfijkaengnhpeoneiagigajhco
ohdihpdgfenligmhnmldmiabdhflokkh, abenhehmjmoifipfpjeaejpbeeihnokp
ackpndpapmikcoklmcbigfgkiemohddk, ceogcehidijhepckebfifkpfogkajdkg
cgijoonmpaboophnagdckdcekmpfokel, dhcfokhhmhenbfmeflifppiedabfggkj
dhcmolikocplmafolinkncghmahimooh, eamjolanjdmgochipodfokkfjaeifhon
efhbachoakbcmbcmfffdgphbpcbldjac, fecipnolpdcmoidbjbnakpjgfikbnaik
gnnagpehbmfalanfjadamobejlldgedo, ijdfpccaiklfhpnamolipbjjijilmhli
khjhfgcimhcnaimdbgjbnbhcojkoceoc, niceocbendibobemckcagggppphheomc
okcfiidnmioajibmhhjpiomgejajiafa, pjjceionkajpednnegoanjjdlhbgkkpc
pjojmkmdealampgchopkfbejihpimjia,

TABLE VIII: Extensions which give access to their storage to specific applications

lpkhcobfjeidpkllbeagkkmmjgbmpfch	mail.google.com
eggdmhdpffgikgakkfojgiledkekfdce	mail.google.com
jmllflbhbembffempimjdbgnaodpoihh	mail.google.com
jmlnhlclbpfcbkaoaegfigepaffoankc	*.google.com,
gaoiiiehelhpkmpkolndijhiogfholcc	netflix.com
ghldlmcbffbcnoofadgcapodmpiimflj	netflix.com
jpgadigdffhcjldfkanacncocacekkie	netflix.com/watch/
peiajekggpiihnhphljoikpjeaahkdcn	beam.pro
bnfboihohdckgijdkplinpflifbbfmhm	plug.dj
aclhfmpoahihmhhacaekgcbjaeojnifa	wordix.io,
hcdfoeppbchkbbpplllggbjkkfokifej	*.vk.com/feed/
hddnlanhlmifafibmlabomkkkobcmchj	thankscoin.org
lhjajgnfmiliphkioedlmbfcdkhdhnkc	*.service-now.com
bmdlalnebjigindhobniianfmhakfelf	robertsspaceindustries.com,
dadggmdmhmfkpglkfpkjdmlendbkehoh	openvideo.droppages.com
pbpfgdgddpnbjcbpofmdanfbbigocklj	tweetdeck-enhancer
ilpkhojfiejdbkgcjbmllngjebdoehim	*.phylotree.org
cfnjeahambijfdljfacldifapdcklhnj	isogg.org
cjkbjhfhpbmnphgbppkbcidpmmbhaifa	*.player.me
ddiaadobgihkgefcaajmkjgmnjakiamn	auth.digitalkeyway.com
dienbdhbgkpddlgaceopelifcjpmkeha	*.gestionderesidencias.es
dnpdkejhfeeipmklhlkdjaoakbkjkkjn	datalane.io
gmjdaaahidcimfaipifeoekglllgdllb	chat.stackexchange.com
kfodnoaejimmmphonklghkimhnhhgbce	overlayBI.com

TABLE IX: Chrome, Firefox and Opera extensions which give web applications access to privileged APIs

Extension unique identifier or name	Web applications to send messages from	Target web applications to access	**Permissions (accessible privileged API)**
		Chrome Browser	
fimckmjeammfdcpldmcigeojkkmeeian	*	*	`eval`, `host`, `storage`, `downloads`
fidaihkgnbcbkkdaoebdionfjenegede	*	*	`eval`, `host`, `storage`
hnkmipajjgbclkombnmigfnpekddlhlh	*	*	`eval`, `host`, `storage`
fajjnmbcianlnhmngmabhgkmgdindlha	*	*	`eval`, `host`, `storage`
efajnkcfjjkcodbhkhaigkffdleomnag	*	*	`eval`, `host`
hoobpdoclliidciecjifpikpnopjpmkh	*	*	`eval`, `host`
kjfjdocojijlledbaanbhpcnkoimghal	*	*	`eval`, `host`
pfofjhnkanlacmgfgjohncmgemffkldl	app.ringostat.com	*	`eval`, `host`, `cookies`, `storage`
gooecknlakggnppmhfpopneedjconjjp	lionlock.com,	*	`eval`, `host`, `storage`
bdiogkcdmlehdjfandmfaibbkkaicppk	*.delfa.com.br	*	`eval`, `host`, `storage`
pgbjjemkcflenaakhiehfdmcdnlnlpbl	www.seejay.cloud	*	`eval`, `host`, `storage`
hdanmfijddamndfaabibmcafmnhhmebi	*.hirogete.com,	*	`eval`, `host`
hpmeebiiihmjelpjmmemlihhcacflflc	*.valleyge.com	*	`eval`, `host`
oejnkhmeilmiplpmenkegjaibnjbappo	search.lilo.org,	*	`eval`, `host`
jkoegdibpkleifbkojmplebjhfllkckn	search.uselilo.org,	*	`eval`, `host`
aopfgjfeiimeioiajeknfidlljpoebgc	*	*	`host`, `cookies`
hlagecmhpppmpfdifmigdglnhcpnohib	*	*	`host`
kpgdinlfgnkbfkmffilkgmeahphehegk	*	*	`host`
bjjpnhdlhpfdebcbhdlmecafnokpjpce	*	*	`host`
bmiedopcajpcehbbfglefijfmmndcaoa	*	*	`host`
jegnjmcegcpodciadcoeneecmkiccfgi	*	*	`host`
jnhibbjmekoijdjaopflcjbjieamifhh	*	*	`host`
jpkfmllgncphdgojhkbcjidgeabaible	*	*	`host`
ilcpdgfepihaomggobhmfiimflngbcoh	starthq.com,	*	`host`, `history`
jpcebpeheognnbogfkpllmmdnimjffdb	mail.google.com,	*	`host`, `management`
cnkgdfnjmgamkcpjdljdncfjcegpgcdg	mail.google.com,	*	`host`, `management`
cfddhmlokgokhcmepddjooekhmgmgfld	*.ok.ru	*	`host`, `downloads`
efhgmgomhamkkmjbgmcpgjnabcfpnaek	*.ok.ru	*	`host`, `downloads`
djhfcchmdelggndcpkgbanfhnpbbijdb	*.ok.ru	*	`host`, `downloads`
fhlkioimlijffnblckmdikkadobdmlgn	*.apistop.com	*	`host`
angncidddapgcmohkdmhidfleomhmfgi	logincat.com,	*	`host`
lndhlcaobijohmgoikmgpgbhepkbhpkl	oneom.tk	*	`host`
olpheomfiimdonpboopcailehdagfhaa	.g3user.com,	*	`host`
idkghekmllmjgnmbohakcddgcclanlca	ln.io	*	`host`
mhdhcccejcjfanablmohbpdbepdkokkj	*.gvt.com.br,	*	`host`
plfffminkgohddbooidppccppgelajfp	mp.weixin.qq.com,	*	`host`
cboekbiaoabkhgjdclenjpipclabkdga	*.apiary.io,	*	`host`
ekeefjfdbaakgbfbagacmckiedkmakem	*.salesmate.io,	mail.google.com,	`host`
lbjbbkhljiimahdeknpckaoiinopofhl	*.appspot.com	mail.google.com,	`host`
ijmbknjhacbaeeoamjajoolgjgdbpkko	*.aliexpress.com,	*.google.com,	`host`
hihakjfhbmlmjdnnhegiciffjplmdhin	mail.google.com	linkedin.com,	`host`
cfbodcmobhpfbjhbennacnanbmpbcfkd	*.aliexpress.com,	appfreaker.com	`host`
ommfijfafanajffiijecdlfjlbgpmgpl	*.treesnetwork.com,	docs.google.com,	`host`
okgfglgogpkomipfflpajohdkaflndoh	ouramazinghome.com	www.google.com	`host`
iiabjaofopjooifoclbpdmffjlgbplod	blog.renren.com	*.github.com	`host`
mcdjehgaflnlmilhefigdkldfdnembhk	*.spotsetter.com	*.amazonaws.com	`host`
lfekjajdgncmkajdpiadkkhhpblngnlc	sub.watch,	zooqle.com,	`host`
gkfpnohhmkonpkkpdbebccbgnajfgpjp	squares.io/fetch,	www.nytimes.com	`host`
pkkbbimilpjmghfhhppamgigileopnkc	*	*	`cookies`
5 Fabasoft extensions (See Table VI)	*	current tab	`cookies`
emiplbkkiabideffmpogkbbogkmofgph	*	-	`downloads`
17 extensions (See Table IV)	vk.com,	-	`downloads`
eadbjnlpeabhbllkljhifinhfelhimha	ok.ru	-	`downloads`
ngegklmoecgejlbkiieccocmpmpmfhim	*.tribecube.com	-	`downloads`
iogibhaacmieogkdgebfbjgoofdlcmgb	*.shutterstock.com	-	`downloads`
ooeealgadmhdnhebkhhbbcmckehpomcj	animevost.org	-	`downloads`
dnohbnpecjinmdpeikpnmheeepnapfci	vtop.vit.ac.in	-	`downloads`
pgmcojeijjhacgkkjaakdafmloncpema	repl.it	-	`downloads`
hacopcfnbokiahlppemnlneooamldola	hypem.com	-	`downloads`
bpkphnbpiagbpinglgejckickdgaghjo	amer...matrix.com	-	`history`
fheihcbdclkdoeadmjfggiamjgkippli	.my-lucky-star.net	-	`topSites`

TABLE IX: Chrome, Firefox and Opera extensions which give web applications access to privileged APIs

Extension unique identifier or name	Web applications to send messages from	Target web applications to access	**Permissions (accessible privileged API)**
		Chrome Browser	
llelondjpcjljnjihdflhpclcpbiaiba	*.msn.com	-	`topSites`
6 Atavi Extensions (See Table V)	atavi.com,	-	`history`, `bookmarks`, `topSites`
31 HD Wallpapers (See Table III)	fliptab.io	-	`history`, `bookmarks`, `management`, `storage`
pnbfclligibfgdknphcodpbcejnkhffp	*	-	`bookmarks`
eihbcgffjehfcgafjljohecmadcefoji	app.launch.menu	-	`bookmarks`
empgohlokhdhhchkenknobacofijiffg	app.launch.menu	-	`bookmarks`
aefmgkhgcmdljpfijlohmbhkhflmbmfi	openoox.com	-	`bookmarks`
dhjhphjhpcelebeaglljbfpipdfkhgi	.azurewebsites.net	-	`bookmarks`
jeabbgpkliknjiacfkfglknajloappkh	yeahap.com,	-	`bookmarks`
22 Extensions (See Table VII)	*	-	`storage`
24 Extensions (See Table VIII)	mail.google.com, ...	-	`storage`
		Firefox Browser	
guretv-ver-tv	*	*	`eval`, `host`, `storage`
buxenger	*	*	`eval`, `host`
bitbucket-server	*	*	`host`
logincataddon	logincat.com,	*	`host`
facebook-photo-zoom-easy	www.facebook.com	*	`host`
facebook-photo-zoom	www.facebook.com	*	`host`
markanabak-eklentisi	*.markanabak.com,	*.wipo.int,	`host`
skimdaddy	*	skimdaddy.com	`host`
the-trees-network	*.treesnetwork.com,	docs.google.com,	`host`
assina-me	*	-	`downloads`
liber-capital	*	-	`downloads`
video-downloader-1	*	-	`downloads`
openvost	animevost.org	-	`downloads`
youtube-video-download-convert	*.youtube.com	-	`downloads`
openvideo	droppages.com	-	`storage`
vgis	*.vonage.com	-	`storage`
		`Opera Browser`	
bmjcngclkmgpfbjcmnbidognkoocpllm	*	*	`eval`, `host`, `storage`
jnmcfakfglphcmgokeeoihifcenjjcgg	*	*	`eval`, `host`
pmpnemphhmmpkcafgpdjanghiaadfbef	*.ok.ru	*	`host`
mpaghnpkgmnikepcgjddhckcedapomkp	*.ok.ru, *.vk.com,	*	`host`
bcabkcaakkjfdlodkolfagbdejhhkigp	*.lazyrobin.ru	*	`host`
bidjmocompdljmeglljcoecikgogfjbb	sub.watch,	zooqle.com,	`host`
aghgmcnoiflhcnfjkckofmjbeinjkena	vk.com,	-	`downloads`
mhjbdafcpnoapkglmldoofhhbpnogehk	vk.com,	-	`downloads`
hajlecmoacenahambneialopbpleihjn	*	-	`storage`
lkdpdiepahdagdknbbjgnadholcdgfib	tweetdeck-enhancer	-	`storage`