Research Article

BEFP: An Extension Recognition System Based on Behavioral and Environmental Fingerprinting

Ting Lyu , Liang Liu , Fangzhou Zhu , Simin Hu , and Renjun Ye

Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210000, China

Correspondence should be addressed to Liang Liu; liangliu@nuaa.edu.cn

Received 9 November 2021; Accepted 31 January 2022; Published 21 February 2022

Academic Editor: Juhriyansyah Dalle

Copyright © 2022 Ting Lyu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Browser extensions are third-party applications that can customize the browsing experience. Previous studies have shown that browser extension fingerprinting can be used to track users and reveal users' privacy information by obtaining the browser extension list. However, the proposal of various defense measures weakens the effectiveness of the existing extension fingerprinting technologies. In this paper, we first propose two extension fingerprinting technologies: JavaScript-based environmental fingerprinting and DOM-based behavioral fingerprinting. They, respectively, capture the operation behaviors of extensions on JavaScript properties and webpage's DOM. Second, we design BEFP, an extension recognition system which comprehensively utilizes the above two technologies to improve the uniqueness of the extension fingerprint. Finally, we collect the latest data set and carry out experiments on the actual scenario where users install multiple extensions. The results show that the true positive rate of extension recognition is as high as 96.3%. And the extension's detectable rate of BEFP is superior to the existing technologies. Moreover, it is proved that the JavaScript-based environmental fingerprinting can complement the DOM-based fingerprinting to distinguish the extensions with the same DOM modification.

1. Introduction

Since the advent of the Internet, various network technologies have been proposed, and the number of users has grown rapidly. Nowadays, people's lives are inseparable from the Internet. All kinds of daily activities are closely associated with Internet applications and mobile Internet applications, in which browsers are playing an increasingly important role. In order to improve the user's browsing experience, the browser allows users to enhance its functionality by installing extensions.

While browser extensions provide convenience [1–3], they also bring many security and privacy threats [4–11]. Once the information about the user-installed extensions is obtained by attackers, the user's identity and privacy may be at risk. Specifically, on the one hand, users install different extensions according to their own needs, thus forming a list of extensions with personal preferences. In this way, the attacker can uniquely fingerprint a user so as to realize the identification of unlogged users, anonymous tracking, and

personalized advertising push. On the other hand, certain browser extensions, such as Trump Filter, which filters content related to former U.S. President Donald Trump, can insinuate the user's political inclination. Users who install the Ya'Muslim extension (which displays prayer times) will reveal their religious beliefs.

For the sake of security, the browser prevents loaded web pages from directly accessing extensions installed by the user. However, recent studies have found that extensions can still be harvested indirectly through browser extension fingerprinting [12–16]. The attacker first creates a honey page (a custom page made up of HTML, CSS, and JavaScript). When a user visits it, his/her browser loads the page and executes its JavaScript code, which automatically collects and generates the browser extension fingerprint by comparing the modifications of extensions to the page and analyzing extensions' WAR (Web Accessible Resources) ID and sends them to the attacker's server. The server matches the fingerprint with the precollected extension fingerprint database to infer extensions installed by the user. The

detailed process is shown in Figure 1. Sjösten et al. [12] first proposed to use the unique and invariant WAR ID to recognize an extension. Subsequently, Starov et al. proposed XHound [17], which utilizes modifications of the extension to the DOM (Document Object Model) of the web page to detect the existence of browser extensions. For example, the advertisement shielding extension deletes ad-related DOM nodes.

To protect the privacy of users, in recent years, many researchers have begun to study the defense schemes against browser extension fingerprinting. Trickel et al. [18] designed the CloakX system that randomizes WAR ID and values of class and ID attributes in DOM. Cloak dramatically weakens the effectiveness of the existing extension fingerprinting technology, thus hindering user privacy inference and extension identification attacks using these technologies.

By analyzing the internal mechanism of typical browser extensions, we found that many extensions modify or rewrite the default properties and functions of the browser's JavaScript engine to achieve certain functions or satisfy security and privacy requirements, resulting in the modification of some accessible property values in the JavaScript runtime. In the case of Canvas Fingerprint Defender, to defend against Canvas-based browser fingerprinting, this extension replaces the function for fingerprint recognition with a custom function, which causes some of the accessible properties of the JavaScript runtime to be changed.

We propose a JavaScript-based environmental fingerprinting technology to distinguish extensions based on their different modifications to JavaScript runtime properties. It invalidates the existing defense techniques to WAR-based and DOM modification-based extension fingerprinting. In addition, for the same kind of extensions, the existing extension fingerprinting technology cannot specifically distinguish them because their DOM modifications are the same. However, our proposed JavaScript-based environmental fingerprinting can recognize such extensions down to the specific extension. In the case of ad-blocking extensions, the prior art can only determine that the user is using an extension for blocking advertisements, while the environmental fingerprinting can specifically infer which extension, such as uBlock Origin, Adblock, etc.

For extensions that do not modify or update JavaScript properties, we design a behavioral fingerprinting technology that recognizes an extension based on its operation behavior (add, delete, modify) on the web page's DOM. Specifically, we only retain the tag name, attribute name, and specific text content in the fingerprint information. This effectively breaks CloakX's defense.

Finally, we comprehensively use the above two technologies to design an efficient browser extension recognition system, BEFP (Behavioral and Environmental Fingerprinting), in order to further improve the uniqueness of extension fingerprint. The experimental results show that BEFP can overcome the defense against the existing extension fingerprinting technologies and recognize extensions with high accuracy.

An earlier version of this article was presented in [26]. In this journal paper, we make the following extensions. We

additionally propose the extension fingerprint recognition technology for multiple extensions. Specifically, we realize the recognition based on the fuzzy matching algorithm for behavioral fingerprints and environmental fingerprints, which effectively improves the accuracy of extension recognition. In addition, we developed the BEFP prototype and collected 91147 extensions on the Chrome Web Store. Then we used random sampling to evaluate the performance of fingerprint recognition. The recognition positive true rate of BEFP is as high as 96.3% and it is better than XHound in terms of detectable rate.

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

- (i) We propose a new extension fingerprinting technology, JavaScript-based environmental fingerprinting, which relies on the difference in the effect of extensions on JavaScript properties and can break the existing extension fingerprint defense schemes.
- (ii) In addition, for extensions that do not modify or update JavaScript properties, behavioral fingerprinting is designed to fingerprint extensions using their behavior on the web page's DOM. It can resist the existing defense measures of DOM modification-based fingerprinting.
- (iii) We evaluate BEFP under the real situation where the user's browser installs multiple extensions. Experimental results show that the presented extension recognition is reached at 96.3% level in true positive rate and 0.7% in false positive rate, respectively.

The rest of this paper is organized as follows: Section 2 mainly expounds on the existing browser extension fingerprinting and defense measures. Section 3 describes two new extension fingerprinting technologies. Section 4 presents the collection of our experimental data set, and we put forward the performance evaluation of extension fingerprinting technologies given in this paper. Finally, the summary of the paper is given in Section 5.

2. Related Work

Traditional user identification technology utilizes state identifiers (such as cookies) to uniquely identify a user. In order to counter this threat, the browser's privacy mode and some extensions (such as Cookie AutoDelete) automatically delete cookies and clear sensitive information at the end of the session. In this way, the effectiveness of the above identification technology is decreased [19]. Mayer [27] first proposed the client can be identified by its underlying operating system and hardware in addition to display settings, browser settings, plug-ins, and add-ons. It is a stateless identification technique and it can compensate for state tracking. Because the technology does not need to store anything on the user's device, it is difficult to detect. Eckert [20] named it browser fingerprinting, and he discovered that fonts and plug-ins were the two most discriminating features, allowing him to uniquely identify

FIGURE 1: Process of browser extension fingerprinting.

94.2% of the surveyed users. Subsequently, fingerprinting technology based on browser extension list, HTML5 Canvas, Audio, and other features gradually attracts the attention of researchers [21].

2.1. Extension Fingerprinting. Chrome extensions are software programs that customize your browsing experience, built on web technologies such as HTML, JavaScript, and CSS [25]. Browser extension fingerprinting is used to uniquely identify a user by obtaining the list of extensions installed by the user. Meanwhile, sensitive information such as religious belief, political inclination, sexual orientation, and age can be inferred based on the extensions installed by the user. Currently, there are two main types of extension fingerprinting technology: WAR-based fingerprinting and DOM modification-based fingerprinting.

2.1.1. WAR-Based Fingerprinting. A WAR refers to an extension's resources that are accessible within the context of a web page. A web page accesses the resources of an extension with the URL format of `chrome-extension://<extensionID>/<resourcePath>`. The `extensionID` is the unique identifier of an extension generated by the Google Web Store and does not change as the extension is updated. The `resourcePath` refers to the path of resources. Sjösten et al. [12] first proposed that extensions could be identified using WAR. It points out that more than half of the top 1,000 Chrome extensions use WARs, which provides a wealth of information for extension detection. In response, Firefox and Trickel et al. [18] came up with the idea of randomizing the WAR ID. However, it is pointed out that, instead of alleviating the problem, the randomized ID will worsen the extension detection problem, allowing the attacker to use them as a reliable fingerprint of the user [13].

2.1.2. DOM Modification-Based Fingerprinting. Extensions are typically implemented by manipulating the page's DOM. An attacker can uniquely identify an extension

by analyzing changes in the page's DOM caused by different extensions. Starov and Nikiforakis [17] developed XHound, the first system which automatically fingerprints browser extensions based on DOM modification. However, CloakX [18] randomizes values of the class and id attributes in the DOM, which confuses extension fingerprints so as to defense extension fingerprinting based on DOM modification.

3. BEFP

This section introduces the extension recognition system we proposed, BEFP, which comprehensively utilizes behavioral fingerprinting based on DOM modification and environmental fingerprinting based on JavaScript to achieve efficient detection and recognition of extensions. Specifically, the victim is induced to visit the attack page by means of a forged website or e-mail. Then the victim's browser loads the page, and extensions installed by the user modify the page. Upon the page full loading, the preinjected script in it will be executed to collect the information of the extensions and send it to the attack server. The attack server matches the information it receives with a prebuilt fingerprint database to recognize the extensions installed by the victim. The architecture of BEFP is shown in Figure 2. BEFP consists of three modules: generation of attack page, construction of extension fingerprint database, and extension recognition.

- (1) Generation of the attack page: extensions installed in the user's browser do not always modify the web page. Most extensions only modify it when the browser loads the web page with a specific domain name or content (e.g., ad-blocking extensions only work when the page contains ads), so we need to design an attack page to trigger the user-installed extension to modify the page as much as possible, thus capturing more extension fingerprint information.
- (2) Construction of the extension fingerprint database: after the extension fingerprint information is captured through the attack page, it is sent to the attack server. In order to recognize the list of user-installed

FIGURE 2: Overview of BEFP.

extensions, we need to collect extension fingerprint offline in advance to build the extension fingerprint database. We have implemented an automatic algorithm that crawls all extensions from Google Web Store. By extracting the differences of the attack page's DOM and JavaScript accessible properties before and after loading the extension, the database can be automatically established.

- (3) Extension recognition: when a user visits the URL of an attack page built by BEFP, the browser loads the page. After it is fully loaded, the user-installed extensions perform operations on the attack page. For instance, Trump Filter filters elements related to Trump in web page's DOM; Canvas Fingerprint Defender replaces a function of browser's JavaScript runtime with its custom function, resulting in some accessible properties of JavaScript runtime to change. At this point, the JavaScript code we pre-injected into the attack page will automatically collect and analyze the modifications of DOM and JavaScript properties and send them to the attack server. The attack server compares the received information with each extension fingerprint in the fingerprint database and finally obtains the list of extensions installed by the user.

3.1. Generation of the Attack Page. We first create an attack page so that users' extension fingerprints can be collected when they visit it. To capture various extension fingerprints and improve the accuracy of extension recognition, the attack page is designed from two levels of the extension's behavior on web page's DOM and JavaScript runtime environment: DOM-Honey Page and JS-Honey Page. The BEFP we proposed makes full use of the advantages of behavioral fingerprinting and environmental fingerprinting, so the above two pages are mixed to get our final attack page.

3.2. DOM-Honey Page. Extensions operate on the DOM in three main ways: delete, add, and modify. (1) Delete operation: Take Adblock as an example; it removes ad-related DOM elements to achieve the blocking of advertising content in webpages; Trump Filter filters all Donald Trump-related content on the page. (2) Add operation: Pinterest is a picture collection extension. Loading the extension will display a collection icon on the picture of the page. This extension enables bookmarking by adding the `<span>` tag to the web page's DOM. (3) Modification operation: The modification involves images, text, and other elements. It generally requires the page to contain specific content to trigger DOM modification. For example, Business to Burrito replaces Business with Burrito. Pics to Kittens replaces all images in the page with images of cats. It is thus clear that triggering extensions requires the presence of some specific HTML elements. To do this, we need to build an attack page which includes as many elements as possible to trigger extensions.

In order to ensure the integrity and comprehensiveness of the attack page, we first conducted experiments with a large number of extensions to analyze the triggering conditions for them to modify the web page's DOM (e.g., the triggering condition of Trump Filter is that the web page contains relevant information about Donald Trump). Specifically, starting from three behaviors: add, delete, and modify, the conditions required for triggering extension to modify web page's DOM are analyzed to generate the attack page.

3.2.1. Extensions Adding New Elements to the Page. Such extensions fall into three main categories:

- (1) HTML element-oriented extensions: this type of extension requires a specific HTML element to be included in the web page before adding a new element to web page's DOM. For example, the Pinterest extension works only on pages that have images, adding a favorite button to the image.

- (2) URL-specific extensions: these extensions are designed for specific URLs. Adblocker for YouTube will only block ads under the `*://*.youtube.com/*` domain name.
- (3) Other extensions: they add DOM nodes to any web page. A user who installs the Evernote extension will add the `<style>` tag to any page he visits.

3.2.2. Extensions Deleting Page Elements. Such extensions require the presence of a specific element in the web page to trigger a DOM deletion operation. For instance, an ad-blocking extension would require webpages to include pop-up ads; extensions like filtering content related to Donald Trump, removing gender identifiers from web pages, and blocking Olympics-related keywords all need webpages to include images, text, and other information that the extension would block.

3.2.3. Extensions Modifying Page Elements. This type of extension modifies the relevant elements of the web page. As previously described, the Business to Burrito extension replaces the text “Business” on the web page. In addition, some extensions are sensitive to tag-specific attribute values like Force Autocomplete. It forces the autocomplete attribute value of the `<input>` tag to be “on,” and only if the autocomplete attribute value is “off” will the DOM modification behavior of the extension be triggered.

Based on the above analysis, we divide extensions operating on the web page’s DOM into the following four categories to capture their fingerprints.

3.2.4. Capturing Extensions That Change Basic HTML Elements. This type of extension modifies basic elements such as images, input boxes, links, etc. To capture these extensions, we create a basic page that contains all the tags and attributes of HTML.

3.2.5. Capturing Extensions That Change the Page Text. Many extensions act on the specified text content of a web page. When the web page contains the specified content, this type of extension will modify or delete it. The text content that triggers diverse extensions is different. For fully capturing the behavior of such extensions, it is necessary to build a text content database that triggers them. How to construct the database is currently an unresolved problem.

We found that the installation page of each extension has an “Overview” option, whose content is mainly divided into the following categories: the basic introduction of the extension, the detailed description of the extension’s functionality, the author of the extension, and the source address of the extension. For extensions that change the page text content, the text triggering the extension is commonly introduced or illustrated in the “Overview.” For instance, the first paragraph in the “Overview” of the Trump Filter reads, “Remove Donald Trump from the Internet.” Therefore, by fetching the “Overview” content of the extension and

analyzing text, we can get the text content of the web page that triggers different extensions, as follows:

First, crawl the first paragraph of each extension’s overview. Second, to remove text redundancy, we preprocess the retrieved text: (1) Convert all content to lowercase. (2) Use regular expressions to filter punctuation marks. (3) Remove terminations by natural language processing. (4) Restore word form and splice. Finally, the trigger text of each extension is captured by comparing the changes of the text content before and after the extension is loaded to form a text content database.

3.2.6. Capturing Extensions That Act on Specific Domains. Such extensions are triggered when the domain name meets specific criteria, such as Adblock for YouTube, which is only effective under the domain name of `*://*.youtube.com/*`. To trigger the operation of such extensions on webpages to grab fingerprints, we embed content from popular websites like YouTube videos and Twitter tweets into the attack page.

3.2.7. Capturing Extensions Modifying Specific Attribute Value. This kind of extension is triggered when the attribute of a HTML element takes a specific value. For example, only when the autocomplete attribute value of the tag `<input>` is “off” can the Force Autocomplete modify web page’s DOM. However, the default value of this attribute is “on.” In order to obtain the fingerprint information of such extensions, it is necessary to get the attributes triggering them and the values of the attributes. To do this, we first extract all attributes with fixed optional values in HTML (for example the autocomplete attribute can only take “on” or “off”) to generate a list of candidate attributes. Then, to find the set of attributes that trigger different extensions, we iterate over each attribute of the list in the crawled text of extensions’ overview. For an attribute that appears in the overview, all its possible attribute values (such as “on” and “off” for the autocomplete attribute above) are added to the attack page.

Through the above process, we complete the establishment of the DOM-Honey Page. Finally, JavaScript code is added to it to automatically grab extensions’ behavioral fingerprint information. The complete process is shown in Figure 3.

3.3. JS-Honey Page. For extensions that do not modify the web page’s DOM, we propose a new extension fingerprinting technology, JavaScript-based environmental fingerprinting, which extracts the extension fingerprint by capturing the modifications of JavaScript properties. Schwarz et al. proposed to identify users’ hardware and software environments by using the differences of JavaScript accessible properties caused by operating systems, CPU architectures, and browser versions, thereby distinguishing users [22].

Inspired by this, consider that loading different extensions on a web page will have an impact on the value of JavaScript accessible properties. For example, FP-Block [23] prevents cross-domain tracking of users by site emitters through assigning different fingerprintable browser

FIGURE 3: Generation of the attack page.

information to different websites. It modifies the information sent to the server. Chrome Zero [24] protects against JavaScript-based microarchitectures and side-channel attacks by dynamically blocking the use of dangerous JavaScript features. To do this, it overrides the related JavaScript function. These behaviors both cause the modification of the accessible properties of JavaScript. We propose an extension fingerprinting technology based on JavaScript runtime environment, which can distinguish extensions that do not modify a web page's DOM according to the differences in modification of JavaScript properties.

To get the JavaScript properties, we need to preconstruct a JavaScript property list comprised of the property name and the corresponding value. JavaScript properties mainly include data properties and accessor properties. Among them, data properties are generally used to store data values directly, which can be accessed directly by JavaScript code. Accessor properties cannot be defined directly, and they must be defined using `Object.defineProperty()`. Therefore, accessor properties cannot be directly obtained by JavaScript.

3.4. Data Properties. The types of data properties mainly include (1) String, Number, Boolean, NULL, Undefined, and Symbol: These basic types of properties have a single simple value. We can read the property name and its corresponding value directly. (2) Array, Object: These types of properties are compound values aggregated from basic types or other objects. We traverse these properties in depth until their value is a basic type of value.

3.4.1. Accessor Properties. The “enumerable” flag inside such properties defaults to false, meaning that they cannot be traversed through a for-in loop. This kind of property is a property in form and a function in essence. The getter function is called when the accessor property is read, so its return value is not a static value but the result of the function. Unlike normal functions, these properties have no arguments. So, we can add a no-argument function like `toString()` to the property list.

Accessible objects such as global objects, functions, HTML DOM, etc. are referenced in the window object or

exist in its child elements. To get as many properties as possible, we traverse all objects under the window object and their corresponding properties with the window object as the root node. These properties include both the object's own properties and enumerable properties on the prototype chain. In order to find a list of properties that are the same for a single extension and different for different extensions, we filter out variable properties such as time-related properties.

Since environmental fingerprinting is primarily concerned with the impact of extensions on browser JavaScript engine; the core of the attack page's construction is how to get the JavaScript property list. For this reason, we choose Honey Page as the basic page and add JavaScript script to dynamically obtain the extension's ID and the property list. See Figure 3 for the detailed process of forming JS-Honey Page.

3.5. Construction of the Extension Fingerprint Database. After the extension fingerprint information captured through the attack page is sent to the attack server, it is necessary to collect the fingerprint of each extension offline in advance so as to identify the user-installed extension list. We build an extension fingerprint database as the comparison database for extension recognition.

In order to improve the efficiency and correctness of the database construction, we design an automatic crawler to obtain all extensions in the Chrome Web Store. Then the browser installs them one by one and loads the DOM-Honey and JS-Honey attack page. The attack page collects the DOM and JavaScript properties modified by each extension and sends them to the attack server, where the original attack page is compared with the modified attack page. Afterward, for each extension, the additions and deletions of the DOM nodes and the JavaScript properties are obtained. They are stored in the behavioral fingerprint database and environmental fingerprint database, respectively. Eventually, the above two databases are integrated to generate a complete extension fingerprint database. The detailed process is shown in Figure 4.

3.5.1. Construction of the Behavioral Fingerprint Database. Many extensions operate on DOM nodes of the web page; e.g., Adblock removes ad-related DOM nodes to filter web

FIGURE 4: Construction of the extension fingerprint database.

ads, and WOT adds DOM nodes to remind the security information about the current page. Regardless of whether DOM changes caused by these extensions are visible to users or not, since the modification is deterministic, the behavioral fingerprint can be extracted via comparing the original web page with the web page modified by the extension to build the behavioral fingerprint database.

However, the construction of the behavioral fingerprint database has faced some new problems and challenges in recent years. With the increasing security awareness to prevent the extension information from being utilized by malicious attackers, many researchers have carried out research on how to defend against extension recognition attacks. Taking CloakX as an example, it confuses the DOM modification by randomizing values of the class and id attribute of DOM elements. Hover Zoom + that helps users zoom images or videos adds the class attribute to certain DOM elements, such as `class="PC-items-item-img img-loaded overzoomlink."` When a user applies CloakX, it randomizes the class attributes, for example, `class="TCMJKLW_56SEPD MTQ37_FGR90 ."` For different extensions of the same user and the same extension of different users, values of the class and id attribute in the web page's DOM are different while using CloakX. As a result, the behavioral fingerprint extracted by the attacker via comparing the original web page with the web page modified by the extension is not deterministic. Eventually, it is unable to accurately identify the list of extensions installed by the user.

To deal with the above problems, we capture DOM changes according to the extension's behavior. Specially, we filter specific attribute values and only keep the tag name, its attributes, and text content. Thereupon, we can effectively eliminate the influence of existing defenses. Based on the above ideas, the construction process of the behavioral fingerprint database is as follows: (1) Crawl the list of all extensions in the Chrome Web Store. (2) According to the preobtained extension list, we use the headless browser to automatically load each extension one by one, visit the DOM-Honey Page, grab the web page's DOM modified by the extension, and send it to the attack server. (3) The attack server compares it with the original DOM of the DOM-Honey Page. (4) Divide the comparison results according to the page tags and filter out the specific value of each attribute in the tags. Only the tag name, attribute name, and text

content are kept. Each tag is added as an array to the behavioral fingerprint database.

The format of the behavioral fingerprint database is shown in Figure 5(a). Each behavioral extension fingerprint is comprised of three parts: the unique extension identifier `war_id`, the added tag set `add_dom`, and the deleted tag set `delete_dom`, where id_1, id_2 are values of `war_id`. `add_dom` and `delete_dom` are collections of tags, each of which includes a tag name and its attributes. For example, a tag named `label1` contains n attributes of `attribute11`, `attribute12`, ..., `attribute1n`. Figure 5(b) shows an example of a behavioral fingerprint for the extension WOT.

3.5.2. Construction of the Environmental Fingerprint Database. This stage aims to collect differences in the JavaScript properties of webpages when installing different extensions. The construction process of the environmental fingerprint database is as follows: (1) According to the preobtained extension list, the headless browser is used to automatically load each extension one by one, visit JS-Honey Page, get the JavaScript properties of the web page, and send them to the attack server. (2) The attack server compares them to the JavaScript properties of the primitive JS-Honey page. (3) The addition and deletion of JavaScript properties obtained by comparison are stored in the environmental fingerprint database.

The format of the environmental fingerprint database is shown in Figure 6(a). Each environmental fingerprint consists of three parts: the unique extension identifier `war_id` and the added and deleted JavaScript property sets: `add_properties`, `delete_properties`. `add_properties` and `delete_properties` are composed of multiple properties, each of which includes the property name and its corresponding value. For example, `add_properties` contains n properties: `property11`, `property12`, ..., `property1n` and their corresponding values are `value11`, `value12`, ..., `value1n`. Figure 6(b) gives an example of the environmental fingerprint for the Rastival extension.

3.6. Extension Recognition. When a potential victim visits our attack page generated in Section 3.1, the fingerprint collection program will judge the page's loading progress according to `document.readyState()`. Then it automatically gets the DOM and JavaScript properties of the web page

```

{
  [
    {
      "war_id" : "id1",
      "add_dom" : [
        "tag1" : {
          "attribute11" : "value11" ,
          "attribute12" : "value12" ,
          .....
          "attribute1n" : "value1n"
        },
        "tag2" : {
          "attribute21" : "value21" ,
          .....
          "attribute2n" : "value2n"
        },
        .....
        "tagm" : {
          "attributem1" : "valuem1" ,
          .....
          "attributemm" : "valuemm"
        },
      ],
      "delete_dom" : [
        "tag1" : {
          "attribute11" : "value11" ,
          .....
        },
        .....
      ]
    },
    {
      "war_id" : "id2",
      "add_dom" : [
        .....
      ]
      "delete_dom" : [
        .....
      ]
    }
  ],
  .....
}

```

(a)

```

{
  [
    {
      "war_id" : "bhmmomiinigofkjcpeggjndpbikblnp",
      "add_dom" : [
        "link" : {
          "href" ,
          "rel"
        },
        "span" : {
          "id"
        },
        "i" : {
          "data-reactroot"
        },
        "div" : {
          "id"
        },
        "div" : {
          "data-reactroot"
        }
      ],
      "delete_dom" : [
        ]
    }
  ]
}

```

(b)

FIGURE 5: Representation of behavioral fingerprints. (a) The format of behavioral fingerprints. (b) An example of behavioral fingerprints.

```

{
  [
    {
      "war_id" : "id1",
      "add_properties" : [
        "property11" : "value11" ,
        "property12" : "value12" ,
        .....
        "property1n" : "value1n" ,
      ],
      "delete_properties" : [
        "property11" : "value11" ,
        "property12" : "value12" ,
        .....
        "property1n" : "value1n" ,
      ]
    },
    {
      "war_id" : "id2",
      "add_properties" : [
        .....
      ]
      "delete_properties" : [
        .....
      ]
    }
  ],
  .....
}

```

(a)

```

{
  [
    {
      "war_id" : "hfhomhacdiijkblnlgnljkldpedoipi",
      "add_properties" : [
        "3" : "3" ,
        "globalThis.3" : "[object Window]" ,
        "globalThis.length" : "4" ,
        .....
        "IDBCursor" :
          "function IDBCursor() { [native code] }" ,
        "History" : "function History() { [native code] }" ,
        .....
        "__proto__.removeEventListener" :
          "function removeEventListener() { [native code] }" ,
        "toLocaleString" :
          "function toLocaleString() { [native code] }"
      ],
      "delete_properties" : [
        "globalThis.length" : "3" ,
        "window.length" : "3" ,
        "self.length" : "3" ,
        "frames.length" : "3" ,
        "length" : "3" ,
        "top.length" : "3" ,
        "parent.length" : "3" ,
        "localStorage.length" : "0"
      ]
    }
  ]
}

```

(b)

FIGURE 6: Representation of environmental fingerprints. (a) The format of environmental fingerprints. (b) An example of environmental fingerprints.

affected by the user-installed extensions and sends them to the attack server.

The attack server compares them with the original attack page (implemented with Python diff lib library) and formats the comparison result to get a quad Q (add_dom, delete_dom, add_properties, delete_properties), where the four elements correspond to added DOM nodes, deleted DOM nodes, added JS properties, and deleted JS properties, respectively.

In actual extension recognition, users typically install multiple extensions rather than a single extension. Therefore, the extension fingerprint we obtained is the result of multiple extensions, which brings a challenge to extension recognition. Specially, the fingerprint jointly produced by multiple extensions cannot be matched exactly one-to-one with a single extension fingerprint in the database. The reason is that extensions have mutual influence and coverage. Taking behavioral fingerprinting as an example, extension “XKCD 1288” replaces the text “Election” with “Eating Contest,” while extension “2016 Election” replaces it with “election.” If the former takes precedence over the latter in actual use, the latter cannot be identified. The same goes for environmental fingerprints. In order to solve these problems, we recognize extensions by utilizing fuzzy matching between Q and the extension fingerprint database collected offline.

More in detail, for each extension fingerprint $\text{FInD} < \text{war\_id}, \text{add\_dom}, \text{delete\_dom}, \text{add\_properties}, \text{delete\_properties} >$ in the database, (1) we propose fuzzyMatch to compare Q and FInD . (2) We judge the comparison result. If the match is successful, then store it in the candidate list candList . (3) For extensions in candList , we filter the extension of which fingerprint is a subset of fingerprints of any other extension in the list. (4) Output the final extension list exList , that is, the user-installed extension list. The recognition process is shown in Algorithm 1.

In the above extension fingerprint matching, there is usually more than one extension installed by the user, that is, the obtained behavioral fingerprint and the environmental fingerprint are the results of the joint action of multiple extensions. Thereby there will be mutual influence among multiple extensions. As a result, we choose the fuzzy matching method to compare the acquired fingerprint with the fingerprints in the database.

As shown in Algorithm 2, FuzzyMatch uses BehaviorMatch and EnvironmentMatch to compare the behavioral fingerprint and environmental fingerprint, respectively. If the two fingerprints reach the matching limit at the same time, then FuzzyMatch returns true. It is considered that the user may have installed the extension and stored it in the candidate list of Algorithm 1.

BehaviorMatch is shown in Algorithm 3. It compares Q and each entry FInD in the extension fingerprint database with the threshold value T ($0 \leq T \leq 1$). As mentioned earlier, $Q[0] \sim Q[3]$, respectively, store captured added/deleted DOM nodes and added/deleted JS properties of the web page. $\text{FInD}[0]$ is the unique identifier of an extension. $\text{FInD}[1] \sim \text{FInD}[4]$ corresponds to $Q[0] \sim Q[3]$ one-to-one. We, respectively, judge the number of elements contained in $Q[0]$

($Q[1]$) in $\text{FInD}[1]$ ($\text{FInD}[2]$): count_+ , count_- . If both meet the threshold requirements, the match is successful. The comparison process of BehaviorMatch and EnvironmentMatch is similar. The only difference is that the matching threshold of the number of web page’s tags is judged in the behavioral fingerprint, while the environmental fingerprint matches the number of JS properties added or deleted. Therefore, we omit the detailed process of EnvironmentMatch here.

4. Evaluation

In this section, we evaluate the performance of the extension recognition system, BEFP. The experimental environment is 4-core Intel Xeon E5-2682 V4 CPU, 8 GB memory, Windows 2016 x64 operating system, and the web browser is Google Chrome 83.0.4103.116. We wrote an automated tool to collect a data set of 91147 extensions in the Chrome Web Store, covering 11 extension categories such as blogging, shopping, developer tools, etc. The experiments are conducted on this data set.

To comprehensively test the performance of BEFP, we designed the following two sets of experiments: (1) Evaluate the detectable rate of extensions with a single extension installed. (2) Test the accuracy of extension recognition in real scenarios when multiple extensions are installed. The evaluation indicators we adopt are TPR (the proportion of correctly identified extensions taking up all installed extensions, namely, the recall rate) and FPR (the percentage of uninstalled extensions predicted as installed).

Furthermore, to evaluate the performance of the environmental fingerprinting, we also conducted an experiment on extensions with the same DOM modification. Finally, the performance of fuzzy matching algorithm used in BEFP is estimated.

4.1. Single-Extension Scenario. Based on the collected 91147 extensions, we installed individual extension on the user’s browser successively and collected the behavioral fingerprints and environmental fingerprints. Then we calculated the detectable rate of extensions. Besides, we selected the experimental data in XHound [17] and CloakX [18] as our comparison objects.

As shown in Figure 7, the detectable rate of XHound [17], CloakX [18], and BEFP is 9.2%, 4.28%, and 17.68%, respectively. It can be seen that BEFP shows great superiority. This result is mainly attributed to (1) our newly proposed environmental fingerprinting, whose detectable extensions account for 73.34% (see Table 1). (2) When building the DOM-Honey Page, we additionally take into account extensions modifying the value of specific attributes, which was not involved in the previous research. It should be noted that since CloakX confuses the WAR-based extension fingerprints by randomizing the WAR ID, here we only compare fingerprints based on DOM modification.

The number of extensions detected by BEFP, behavioral fingerprinting, and environmental fingerprinting is shown in Table 1. The results show that 16166 extensions (17.68% of

Input: Q : extension fingerprint to be recognized, DB : extension fingerprint database, T_1, T_2 : matching thresholds of behavioral fingerprints and environmental fingerprints.

Output: $exList$: list of identified user-installed extensions.

```

(1) for  $FInD$  in  $DB$  do
(2)   result = fuzzyMatch ( $FInD$ ,  $Q$ ,  $T_1$ ,  $T_2$ )
(3)   if result == true then
(4)      $exList.append$  ( $FInD$ )
(5)  $candList = exList$ 
(6) for  $i$  in  $exList$  do
(7)   for  $j$  in  $candList$ ;  $j \neq i$  do
(8)     if  $i < j$  then
(9)       delete  $i$  from  $exList$ 
(10)    break
(11) return  $exList$ .
```

ALGORITHM 1: Extension recognition.

Input: Q : extension fingerprint to be recognized, $FInD$: an entry in extension fingerprint database, T_1, T_2 : matching thresholds of behavior fingerprints and environment fingerprints, respectively.

Output: result: matching result.

```

(1) result1 = BehaviorMatch ( $FInD$ ,  $Q$ ,  $T_1$ );
(2) result2 = EnvironmentMatch ( $FInD$ ,  $Q$ ,  $T_2$ );
(3) if (result1 == true)^(result2 == true) then
(4)   result = true
(5) else
(6)   result = false
(7) return result.
```

ALGORITHM 2: Fuzzy matching.

Input: Q : extension fingerprint to be recognized, $FInD$: an entry in extension fingerprint database, T : matching threshold, $count_+$, $count_-$: number of matching fingerprint elements.

Output: $exList$: list of identified user-installed extensions.

```

(1)  $count_+ = 0$ ,  $count_- = 0$ 
(2) for  $i$  in  $FInD[1]$  do
(3)   if  $i < Q[0]$  then
(4)      $count_+ ++$ 
(5) for  $j$  in  $FInD[2]$  do
(6)   if  $j < Q[1]$  then
(7)      $count_- ++$ 
(8) if ( $count_+ \geq round(count(FInD[1] * T))$ )^( $count_- \geq round(count(FInD[2] * T))$ ) then
(9)   result = true
(10) else
(11)   result = false
(12) return result.
```

ALGORITHM 3: BehaviorMatch.

the dataset) can be detected by BEFP, of which 4310 extensions only have behavioral fingerprints, 10361 extensions only have environmental fingerprints, and 1495 extensions have both fingerprints.

As mentioned in Section 3.2, to prevent extension information from being used by malicious attackers, some defense schemes against extension recognition have been proposed in recent years. For example, CloakX confuses the

DOM fingerprint information of different extensions by randomizing values of the class and id attribute of DOM elements, thus weakening the attack power. To eliminate the influence of the defense, the behavioral fingerprint we proposed ignores the attribute value. Nevertheless, there are still 5805 extensions (accounting for 6.37% of the data set) that can be detected, among which 687 extensions are capable of modifying or deleting the text content of the web

FIGURE 7: The detectable rate of XHound, CloakX, and BEFP.

TABLE 1: Statistics of the number of detectable extensions.

Attack	Detectable	Uniqueness
Behavioral fingerprinting	5805	4310
Environmental fingerprinting	11856	10361
BEFP	16166	—

page, and 3907 extensions can modify the DOM in any website without any triggering condition. On the other hand, the JavaScript-based environmental fingerprinting shows strong superiority, 11856 extensions (13.0% of the data set) can be detected, and 10361 extensions (11.37% of the data set) can only be uniquely detected by it.

4.2. Multiple Extensions' Scenario. Existing extension fingerprinting techniques are all oriented towards a single extension; that is, the attack page only loads a single extension at a time. In practice, users often install multiple browser extensions. To the best of our knowledge, we present the first extension recognition for multiple extensions, BEFP.

We install multiple extensions (referred to as extension combination later in this paper) on the user's browser in turn and collect the behavioral and environmental fingerprint of the extension combination. Then it is compared with the fingerprint in our prebuilt extension fingerprint database, and the accuracy of BEFP's extension recognition is calculated according to the comparison results.

The detailed process is as follows. First, we randomly selected 100 samples from extensions with both behavioral fingerprint and environmental fingerprint as the data set. Then, we generate extension combination sets ES_i ($2 \leq i \leq N$), where $ES_i = \{g_{ij} \mid g_{ij} \in ES_i, 1 \leq j \leq m\}$, and g_{ij} is an extension combination including i extensions. In the experiments, $N = 10$, $m = 20$. After that, for $\forall 2 \leq i \leq N$, $\forall g_{ij} \in ES_i$, install extensions according to g_{ij} and visit our attack page. The fuzzy matching algorithm is used to compare the fingerprint of g_{ij} with fingerprints in database.

It should be emphasized that the extensions selected here have two kinds of fingerprints: behavioral and environmental. Because we cannot detect the presence of an undetectable extension, it does not interfere with other extensions in any way

to affect the detection results. So, the undetectable extensions are helpless in testing the effectiveness of our system and can artificially increase TPR and FPR. Therefore, this experiment can effectively evaluate the reliability of BEFP in practical use.

Table 2 gives the evaluation results. The recognition accuracy and false alarm rates are $TPR = TP / (TP + FN)$ and $FPR = FP / (FP + TN)$, respectively. TP refers to the correctly detected installed extensions, TN means the correctly judged extensions that are not installed, FP represents the uninstalled extensions incorrectly detected as installed, and FN is the installed extensions that are not detected. The higher the value of TPR (closer to 1) and the lower the value of FPR (closer to 0), the better the effect of BEFP under multiple extension scenarios.

As can be seen from the table, the accuracy of BEFP remains stable at around 95%, with the highest accuracy up to 96.3%. The false positive rate increases with the increase of the number of installed extensions. Although BEFP filters the candidate list obtained by matching, as the number of installed extensions increases, the fingerprints of extension combinations become more complex. It makes some extension fingerprints with simple structure wrongly identified as installed extensions, thus increasing the FPR.

In particular, the 100 extensions used in the experiment have both behavioral fingerprints and environmental fingerprints, which reduce the TPR and FPR of fingerprint matching. For example, for the five extensions numbered 1–5, extensions numbered 1–7 are matched by the behavioral fingerprint alone, and the environmental fingerprint alone is matched by the six extensions numbered 1–4, 7, and 8. However, considering both extension fingerprints together, we get five extensions numbered 1–4 and 7. Actually, for partial extension, we can only obtain one of the above two kinds of extension fingerprints, so the experimental accuracy in Table 2 is a lower bound.

4.3. Recognizing Extensions with the Same Functionality.

Extensions with the same function have similar behaviors to the web page's DOM. Their operations of deleting, modifying, and adding are basically the same. For example, ad-blocking extensions filter web ads by removing ads-related DOM, so it is difficult to distinguish extensions with the same function using DOM-based behavioral fingerprinting. To solve this problem, JavaScript-based environmental fingerprinting we propose can be applied to identify these extensions.

To evaluate its performance, we conducted an attack experiment on extensions with the same DOM modification. Due to space limitation, only the experimental results of ad-blocking extensions are given. As shown in Table 3, three groups of ad-blocking extensions are constructed. The first group includes Adblock, Adblock Plus, Adblock Unblock, and Adblock. The second group includes Adtrooper Adblocker and A-blocker. The third group includes uBlocker and Hola ad remover. The extension behavioral fingerprints within the same group are the same.

As shown in Table 3, three groups of ad-blocking extensions with the same behavioral fingerprints have different effects on JavaScript properties. These extensions can be

TABLE 2: The accuracy of BEFP in the multiple extensions' scenario.

	2	3	4	5	6	7	8	9	10
TPR (%)	95.0	95.0	96.3	95.0	95.8	95.0	94.4	95.0	95.0
FPR (%)	0.7	2.2	2.1	4.5	3.6	4.6	6.6	7.1	6.9

TABLE 3: The accuracy of BEFP in the multiple extensions' scenario.

Group number	Extension	Add properties	Delete properties
1-1	AdBlock	2	14
1-2	Adblock Plus	2	2
1-3	Adblock Unblock	0	0
1-4	Adblock	6	3
2-1	Adtrooper Adblocker	2	2
2-2	A-blocker	2	8
3-1	uBlocker	3	3
3-2	Hola ad remover	0	6

```

{
  [
    "war_id" : "ppdonaappkjkbgbncmmjencphdclioab",
    "add_properties" : [
      "webkitRTCPeerConnection" : "function () { [native code] }" ,
      "WebSocket" : "function () { [native code] }" ,
      "RTCPeerConnection" : "function () { [native code] }"
    ],
    "delete_properties" : [
      "webkitRTCPeerConnection" : "function RTCPeerConnection() { [native code] }" ,
      "WebSocket" : "function WebSocket() { [native code] }" ,
      "RTCPeerConnection" : "function RTCPeerConnection() { [native code] }"
    ]
  ]
}

{
  [
    "war_id" : "lalfpjdbhpmnhfofkckdpkljeilmogfl",
    "add_properties" : [],
    "delete_properties" : [
      "globalThis.2" : "[object Window]" ,
      "window.2" : "[object Window]" ,
      "self.2" : "[object Window]" ,
      "frames.2" : "[object Window]" ,
      "top.2" : "[object Window]" ,
      "parent.2" : "[object Window]" ,
    ]
  ]
}

```

FIGURE 8: Environmental fingerprints of uBlocker and Hola ad remover.

accurately identified even if only a number of JavaScript property changes are extracted as features. Moreover, the JavaScript properties modified and their values are different, which can be used to further enhance the unique recognition of the extension. The environmental fingerprints of uBlocker and Hola ad remover are shown in Figure 8.

In summary, although the DOM changes of extensions with the same functionality are basically the same, the JavaScript-based environmental fingerprinting we proposed can be used to accurately distinguish these extensions.

4.4. Threshold of Fingerprint Matching. Since there is usually more than one extension installed by users and they may influence and cover each other, we propose a fuzzy matching algorithm to identify each extension.

The selection of the matching threshold parameter is the key problem in the fuzzy matching process. If the threshold is too high, part of information in extension fingerprint will be lost due to the mutual influence of extensions. What is more, the number of matched extensions will decrease, thus reducing the recognition accuracy. If the threshold is too low, the extension without many DOM modifications will be misjudged as the installed extension, resulting in high false positive rate.

We used the same data set and experimental settings in Section 4.2. To ensure the accuracy of the experiment, the matching thresholds of behavioral fingerprint and environmental fingerprint were tested, respectively. A mismatch interval of 10%~30% is allowed. The results are shown in Figures 9 and 10.

As can be seen from Figure 9, with the decrease of the matching threshold, the identification accuracy of the

FIGURE 9: The accuracy of behavioral fingerprinting under different matching thresholds.

FIGURE 10: The accuracy of environmental fingerprinting under different matching thresholds.

behavioral fingerprint increases, while FPR increases significantly. For the environmental fingerprinting shown in Figure 10, when the matching threshold is decreased, TPR is greatly improved, while the false positive rate does not change much. According to the experimental results, when the matching threshold of behavioral fingerprint and environmental fingerprint is 90% and 70%, respectively, the performance of BEFP is better.

In addition, it can be observed that when the number of installed extensions increases, the TPR decreases to some

extent, and FPR increases accordingly. For TPR, with the increase of the number of extensions, the degree of mutual influences between extensions also increases, resulting in coverage and loss of fingerprint information. This conclusion can also be verified by Figure 9. When the matching threshold is lowered, TPR is also alleviated and FPR is increased. The reason is that the more the extensions a user installs, the richer the fingerprint information an attacker can capture and the more likely some extensions with short fingerprints are to be false positives.

5. Conclusions

With the emergence of various defense measures, the extension fingerprinting technologies based on WAR and DOM cannot accurately identify the user-installed extension list. Firstly, we propose a novel extension fingerprinting technology, JavaScript-based environmental fingerprinting, which utilizes the inconsistency caused by the different effect of extensions on JavaScript properties. And it solves the problem that DOM-based fingerprinting can not distinguish the extensions with the same DOM modification.

Secondly, we propose a behavioral fingerprinting technology based on the DOM modification to the web page, which effectively avoids the confusion caused by the existing defense measures of DOM-based fingerprinting. Finally, in order to improve the uniqueness of extension fingerprints, we design BEFP which combines the above two fingerprinting technologies. And we carried out experiments on the actual scenario where users install multiple extensions. The results show that BEFP can achieve 96.3% true positive rate, and it is better than XHound in terms of detectable rate.

People install and uninstall extensions all the time, and some might not have any extensions installed to begin with. In order to accurately identify users, it is very important to update and maintain the extension fingerprint database. This will serve as a direction for us to improve BEFP in the future. In addition, for different browsers, the list of user-installed extensions and the methods of extension detection are different. The research on fingerprinting technology of other browser extensions is one of our main works in the future.

Data Availability

Publicly available datasets were analyzed in this study. These data can be found at <https://github.com/NUAAlvting/befp>.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This research was funded by the National Natural Science Foundation of China under Grant no. U20B2050 and the Science and Technology Funds from National State Grid Ltd. (The Research on Key Technologies of Distributed Parallel Database Storage and Processing Based on Big Data).

References

- [1] U. Iqbal, Z. Shafiq, and Z. Qian, "The ad wars: retrospective measurement and analysis of anti-adblock filter lists," in *Proceedings of the 2017 Internet Measurement Conference*, pp. 171–183, London United Kingdom, November 2017.
- [2] A. Mathur, J. Vitak, A. Narayanan, and M. Chetty, "Characterizing the use of browser-based blocking extensions to prevent online tracking," in *Proceedings of the 2017 Fourteenth Symposium on Usable Privacy and Security*, pp. 103–116, Baltimore, MD, USA, August 2018.
- [3] P. Snyder, C. Taylor, and C. Kanich, "Most websites don't need to vibrate: a cost-benefit approach to improving browser security," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 179–194, Dallas, TX, USA, October 2017.
- [4] D. F. Somé, "Empoweb: empowering web applications with browser extensions," in *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP)*, pp. 227–245, IEEE, San Francisco, CA, USA, May 2019.
- [5] S. Calzavara, M. Bugliesi, S. Crafa, and E. Steffanlongo, *Fine-grained Detection of Privilege Escalation Attacks on Browser Extensions*. European Symposium on Programming Languages and Systems, Springer, Berlin, Germany, pp. 510–534, 2015.
- [6] M. Ter Louw, J. S. Lim, and V. N. Venkatakrishnan, "Extensible web browser security," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Berlin, Germany, pp. 1–19, 2007.
- [7] A. Barth, A. P. Felt, P. Saxena, and A. Boodman, *Protecting Browsers from Extension Vulnerabilities*, 2010.
- [8] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, "VEX: Vetting browser extensions for security vulnerabilities," *USENIX Security Symposium*, vol. 10, pp. 339–354, 2010.
- [9] N. Carlini, A. P. Felt, and D. Wagner, "An evaluation of the google chrome extension security architecture," in *Proceedings of the 21st USENIX Security Symposium*, pp. 97–111, USENIX Security 12, Boston, MA, USA, August 2012.
- [10] K. Onarlioglu, M. Battal, W. Robertson, and E. Kirda, "Securing legacy firefox extensions with SENTINEL," *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, Berlin, Germany, pp. 122–138, 2013.
- [11] K. Onarlioglu, A. S. Buyukkayhan, W. Robertson, and E. Kirda, "Sentinel: Securing legacy firefox extensions," *Computers & Security*, vol. 49, pp. 147–161, 2015.
- [12] A. Sjösten, S. Van Acker, and A. Sabelfeld, "Discovering browser extensions via web accessible resources," *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 329–336, Scottsdale, AZ, USA, 2017.
- [13] A. Sjösten, S. Van Acker, P. Picazo-Sanchez, and A. Sabelfeld, "Latex gloves: Protecting browser extensions from probing and revelation attacks," *Power*, vol. 57, 2018.
- [14] G. G. Gulyas, D. F. Some, N. Bielova, and C. Castelluccia, "To extend or not to extend: on the uniqueness of browser extensions and web logins," in *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, pp. 14–27, Toronto Canada, October 2018.
- [15] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Extension breakdown: security analysis of browsers extension resources control policies," in *Proceedings of the 26th USENIX Security Symposium*, pp. 679–694, USENIX Security 17, Vancouver, BC, Canada, May-2017.
- [16] U. Fiore, A. Castiglione, A. De Santis, and F. Palmieri, "Countering browser fingerprinting techniques: Constructing a fake profile with google chrome," in *Proceedings of the 2014 17th International Conference on Network-Based Information Systems*, pp. 355–360, IEEE, Salerno, Italy, September 2014.
- [17] O. Starov and N. Nikiforakis, "XHound: quantifying the fingerprintability of browser extensions," in *Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP)*, pp. 941–956, IEEE, San Jose, CA, USA, MAY2017.
- [18] E. Trickel, O. Starov, A. Kapravelos, N. Nikiforakis, and A. Doupe, "Everyone is different: client-side diversification for defending against extension fingerprinting," in *Proceedings of the 28th USENIX Security Symposium*, pp. 1679–1696, USENIX Security 19, Santa Clara, CA, USA, August 2019.

- [19] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy, “Fp-scanner: the privacy implications of browser fingerprint inconsistencies,” in *Proceedings of the 27th USENIX Security Symposium*, pp. 135–150, USENIX Security 18, Baltimore, MD, USA, August 2018.
- [20] P. Eckersley, “How unique is your web browser?” *International Symposium on Privacy Enhancing Technologies Symposium*, Springer, Berlin, Germany, pp. 1–18, 2010.
- [21] S. Wu, S. Li, Y. Cao, and N. Wang, “Rendered private: making GLSL execution uniform to prevent WebGL-based browser fingerprinting,” in *Proceedings of the 28th USENIX Security Symposium*, pp. 1645–1660, USENIX Security 19, Santa Clara CA USA, August 2019.
- [22] M. Schwarz, F. Lackner, and D. Gruss, *JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits*, NDSS, 2019.
- [23] C. F. Torres, H. Jonker, and S. Mauw, “FP-Block: useable web privacy by controlling browser fingerprinting,” *European Symposium on Research in Computer Security*, Springer, Berlin, Germany, pp. 3–19, 2015.
- [24] M. Schwarz, M. Lipp, and D. Gruss, “JavaScript zero: real JavaScript and zero side-channel attacks,” *NDSS*, vol. 18, p. 12, 2018.
- [25] Overview of Chrome extensions, <https://developer.chrome.com/docs/extensions/mv3/overview/> (30 December 2020), 2021.
- [26] T. Lyu, L. Liu, F. Zhu, J. Yang, S. Hu, and Y. Huang, *NEEX: An Automated and Efficient Tool for Detecting Browser Extension Fingerprint*, EISA2021 (accepted), 2021.
- [27] J. R. Mayer, *Any Person. . . a Pamphleteer*: *Internet Anonymity in the Age of Web 2.0*, Undergraduate Senior Thesis, p. 85, Princeton University, Princeton, NJ, USA, Princeton University, 2009.