Fingerprinting Web Browser for Tracing Anonymous Web Attackers

Xiaofeng Liu
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
leoxf820@gmail.com

Qixu Liu
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China
liuqixu@iie.ac.cn

Xiaoxi Wang
Beijing University of Technology
Beijing, China
destinyxixi@gmail.com

Zhaopeng Jia
Beijing University of Posts and Telecommunications
Beijing, China
jia_zp@bupt.edu.cn

***Abstract*—As web attackers hide themselves by using multi-step springboard (e.g., VPN, encrypted proxy) or anonymous network (i.e. Tor network), it raises a big obstacle for traceability and forensics. Furthermore, traditional forensics methods based on traffic and log analysis are just useful for analyzing attack events but useless for fingerprinting an attacker. Because of this, the browser fingerprinting technique which makes use of slight differences among different browsers was come up with. However, although this technique is effective for tracing attackers, countermeasures have been proposed, such as blocking extensions, spoofing extensions and *Blink* (a dynamic reconfiguration tool). These countermeasures will lead to changes of fingerprints.**

To solve the instability of browser fingerprints, we present an enhanced solution aiming at tracing attackers continuously even if the fingerprint changes within a particular period of time. By introducing secondary attributes, employing browser storage mechanisms and designing correlation algorithms, we implement the prototype system to examine the accuracy of our approach. Experimental results show that our proposed solution has the ability to associate different fingerprints from a single platform and the accuracy of tracing anonymous web attackers increases by 24.5% than traditional fingerprinting techniques.

Keywords—fingerprint; tracing; correlation algorithm

I. INTRODUCTION

It is known that browser cookies[1] are often used to keep the server side sessions by storing the corresponding key (usually defined as *sessionid*) to solve the problem that the HTTP protocol is stateless[2]. In addition, they are also used to trace users in some news portals or e-commerce sites by setting the values which represent the user's favorite news sections or the goods in the shopping cart into the cookie keys. In a sense, it increases the convenience, on the contrary, it creates a privacy issue. Although modern browsers provide configuration options to disable cookies, few people know or use them properly. Actually, a new technique called browser fingerprinting has been adopted gradually, which makes good use of the slight differences among different browsers. Eckersley[3] presents a method that through the collection of a set of attributes (e.g., platform, plugins, fonts, screen resolution, color depth), it is possible to generate a browser fingerprint that has an extremely low collision rate to trace users. These attributes can be obtained via JavaScript directly or indirectly. Naturally, the browser fingerprint technique appears to be an excellent solution for non-cookie tracing.

However, this method has an obvious disadvantage that the attributes on which the final fingerprint depends are unstable. In other words, these attributes can be easily changed when users modify the related parameters actively or do some operations unconsciously or passively such as updating the browser, installing a plugin. Furthermore, another disadvantage is that even if one attribute changes, the final fingerprint can be totally different and there are no secondary attributes which will help to determine the correlation between two different fingerprints. These two disadvantages make the effectiveness of this tracing technique reduce.

In this paper, we propose an improved technique to establish a more complete tracing solution which is based on the browser fingerprinting and enhanced by adding several significant improvements. These improvements effectively solve the problem of the instability of fingerprints. We aim at tracing a visitor continuously in spite of the change of his fingerprint within a particular period of time. To achieve this goal, we first introduce several secondary attributes into our approach. Some of these attributes play the decisive role in determining whether a fingerprint's origin is the same as the previous one. Secondly, we take advantage of the browser storage mechanisms (e.g., standard http cookie, flash local shared objects, HTML5 local storage) as a countermeasure since someone may change the fingerprint frequently to get rid of tracing. Finally, we propose a correlation algorithm and define a similarity metric to compare two different fingerprints. The calculation method of the metric will take consideration of not only the attributes which are used to generate fingerprints, but also the secondary attributes we introduce and the details of particular attributes (e.g., plugins and fonts).

We implement the experimental system to determine the accuracy of the fingerprint relevance calculated by our approach. We examine our system's ability that whether it can judge of different fingerprints from a single origin when people modify one or more attributes on purpose. We also test if using third-party browser plugins which have the functions to change browser objects can escape the trace of our system. The results show that our approach of enhanced fingerprint tracing has a high rate of accuracy of correlation judging.

978-1-5090-1192-6/16 $31.00 © 2016 IEEE
DOI 10.1109/DSC.2016.78

The **main contributions** of this paper are:

- Introducing secondary attributes that are helpful for correlation judging but not easy to be changed.
- Taking full use of browser storage mechanisms for fingerprint's persistence in browsers.
- A similarity metric that measures the correlation between two different fingerprints.

Section II presents the background and related work on browser fingerprinting. Section III describes our enhanced solution, including secondary attributes, the fingerprint's persistence with browser storage mechanisms, and discusses our fingerprint correlation analysis model. Section IV presents the implementation of our prototype system. Section V shows the experimental results. Section VI concludes this paper.

II. BACKGROUND

Eckersley[3] implements the browser fingerprinting algorithm by collecting a commonly and less-commonly known characteristics as measurements. Some of these measurements are indicated in TABLE I. On their website, they declare that the browser fingerprint appears to be unique among more than six million fingerprints they tested. In this section, we review current techniques of fingerprinting and countering fingerprint-based tracing.

A. Current fingerprinting techniques

In Eckersley's early paper, it demonstrates that eight separate strings are used to constitute a fingerprint as is presented in TABLE I. Some of these attributes (e.g., User Agent, HTTP ACCEPT headers) are sent to web servers through static HTTP requests; others such as screen resolution and plugins are collected and asynchronously posted via JavaScript, so called AJAX. However, as Eckersley says, they did not implement tests for browser storage mechanisms, which is what we hope to improve in this paper. FingerprintJS, an open source project on Github based on Eckersley's research, extends the measurements, such as Canvas and WebGL. The Canvas is a new characteristic of HTML5 standard which allows for dynamic, scriptable rendering of 2D shapes and bitmap images. Canvas fingerprinting was first proposed by Keaton Mowery and Hovav Shacham[5] in 2012. Due to the character that different browsers and hardware lead to slight differences of the image Canvas rendered, this technique is suitable for user tracing. The mechanism of WebGL fingerprinting is similar to HTML5 Canvas. The *navigator.maxTouchPoints* object is also introduced as an attribute which is adopted to test for touch capable hardware or multi-touch capable hardware.

B. Employing the fingerprinting

Up to now, a lot of websites employ fingerprinting techniques to trace their users, as well as common cookies. For instance, Amazon states they will "collect and analyze the information such as browser type, version, and time zone setting, browser plugin-in types and versions, operating system, and platform" in the privacy notice. Similarly, Yahoo states they automatically receive and record information (IP address, software and hardware attributes). In Google Privacy Policy, it indicates they collect device-specific information including unique device identifiers. Also third-party commercial corporations provide more professional services for user tracing, such as BlueCava and ThreatMetrix.

TABLE I.

EXAMPLE OF A BROWSER FINGERPRINT

Browser Characteristic	Value (Firefox)
User Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:44.0) Gecko/20100101 Firefox/44.0
Platform	Win32
Language	zh-CN
Screen Size and Color Depth	1920x1080x24
Time Zone	-480
DNT Header Enabled?	FALSE
Hash of canvas fingerprint	bc5fba5d6d87d41e203a2e76d024092c
Browser Plugin Details	Plugin 0: China Online Banking Assistant; COBA Plugin DLL; npcoba32.dll; (; application/coba; *). Plugin 1: Foxit PhantomPDF Plugin for Mozilla; Foxit PhantomPDF Plug-In For Firefox and Netscape; npFoxitPhantomPDFPlugin.dll;
System Fonts	Arial, Arial Black, Calibri, Cambria, Cambria Math, Comic Sans MS, Consolas, Courier, Courier New, Georgia, Helvetica, Impact, Lucida Console, Lucida Sans Unicode, ..., Verdana, Wingdings, Wingdings 2, Wingdings 3

C. Existing solutions for countering fingerprint-based tracing

We now illustrate the existing incomplete solutions which, to some degrees, limit the effectiveness and accuracy of browser fingerprinting. However, these solutions have some problems or shortcomings more or less.

1) Limit to executing JavaScript: Custom options are available in modern browsers to satisfy the individual needs of users, including the security requirements. Internet Explorer allows users to modify security options, for example, banning JavaScript and Flash. Firefox also provide a configuration parameter (i.e. *javascript.enabled*) to turn on or off JavaScript parsing engine. Some third-party blocking extensions, such as NoScript and Privacy Badger, can also limit scripts loading. NoScript is a free and open-source extension for Mozilla-based web browsers, which allows JavaScript only if the host is in the whitelist or considered trusted by its users. Privacy Badger is the official extension of Electronic Frontier Foundation[1] and has the ability of discovering and blocking tracing scripts. However, this is not a good method for countering browser fingerprint tracing, for modern websites make heavy use of front-end techniques, simply disabling scripts will negatively impact visitors' experience. Worse, adopting block extensions even increases the identifying information for fingerprinting.

2) Using spoofing extensions: Some third-party extensions allow users to modify part of browser's parameter values, leading to changes of final fingerprints. *User Agent Switcher* is a popular extension in Firefox, which can easily switch user agent to disguise as other browsers or other version of Firefox. *Theme Font & Size Changer* lets user change the global font size and font family used in Firefox, which will affect the Canvas fingerprint results. Although this method can change fingerprint on purpose, more often it is contradictory. For

[1]https://www.eff.org/

instance, a user only switches the user agent to an iPad device by a spoofing extension while the value of navigator.platform object is MacIntel. So, this single attribute's spoofing will easily fail when confronting the complex correlation arithmetic, cross-browser fingerprinting techniques or browser storage mechanisms.

3) Using Tor browser: The Tor browser is part of Tor anonymous network[6][7] and is modified from Firefox. As Tor browsers have a unique configuration, it appears to be hard to use fingerprinting to trace them. However, because the normalization of configuration cannot satisfy all Tor users' requirements, a small percentage of users may employ custom settings or install additional plugins, leading to be traceable again.

4) Blink: Blink is a solution proposed by Pierre Laperdrix[4] to defend against browser fingerprint tracing. It regularly changes the four most distinctive elements in a fingerprint: the operating system, the browser, the list of fonts and the list of plugins, as is called Diversified Platform Components or DPC. Blink creates DPC configurations by randomly selecting components from a large pool, called the diversity reservoir, and then assembles these components and reconfigures the browsing platforms, leading to diverse fingerprints being exhibited over time. However, Blink only considers changing the attributes dynamically and regularly while it ignores browser storage mechanisms. Browser storage can recover the previous fingerprint stored in the local host no matter what the current fingerprint is. And the attributes Blink doesn't cover will also help to trace visitors.

III. METHODOLOGY

In this section, we present our improved solutions of browser fingerprinting. On browser-side, we explain what secondary attributes we introduce and what browser storage mechanisms we adopt. We also elaborate on the fingerprint correlation analysis model (Fig. 1) we establish and discuss its reasonability.

A. Secondary attributes

The attributes which traditional browser fingerprinting employs as measurements are commonly BOM (i.e. Browser Object Model) objects, such as *navigator* and *screen*. The instability of these attributes weakens the effectiveness of fingerprint tracing. Using more stable attributes is one of the solutions we consider.

1) Intranet IP address: As is known to all, IP address is a numerical label assigned to each device on the Internet. Can we use a browser fingerprint together with a public IP address to determine the uniqueness of a user? Certainly not. Because the IP reuse technique (e.g., NAT) convert different intranet IPs to a single public IP. However, if we get a visitor's intranet IP address, it will provide a powerful reference for tracing. Although there is no APIs to get intranet IP via JavaScript in ECMAScript standard, we draw attention to another way called WebRTC leak. WebRTC is an open-source

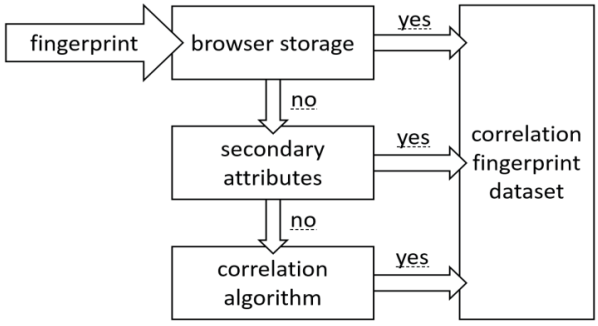

Fig. 1. Fingerprint correlation analysis model.

standard that enables the browsers to make voice or video calls without needing any plugins. WebRTC[8] allows requests to be made to STUN (Session Traversal Utilities for NAT) servers which return the "hidden" home IP-address as well as local network addresses for the system that is being used by the user. And these results of the requests can be accessed via JavaScript, as follows:

|10.10.20.48|169.254.165.62|169.254.0.227|192.168.253.1|

In most cases, a user's network environment is stable, so the results of intranet IPs are referable for fingerprinting. The advantage of employing WebRTC leak is that the results are accurate even though a user connects the Internet via VPN or a proxy, and they cannot be easily faked as well.

2) ActiveXObject: ActiveX is a loosely defined set of technologies developed for Internet Explorer by Microsoft for sharing information among different applications. Because ActiveX is only available for IE, it can be used to determine whether a browser is Internet Explorer or not. And the ActiveXObject is powerful for developers to access significant information (e.g., computer name, domain, user name, disk, processer, applications) of the platform, which makes it possible to fingerprint not only the browser but also the host. To determine the sub-version of the applications installed in the system, we can create different instances of ActiveXObject and know it according to the types of returned values and the exception we catch, as follows:

```
try {
    ma = new ActiveXObject ("SharePoint.OpenDocuments.4")
} catch (e) {}
try {
    mb = new ActiveXObject ("SharePoint.OpenDocuments.3")
} catch (e) {}
if ((typeof ma) == "object" && (typeof mb) == "object")
{
    version = "Office 2010"
}
if ((typeof ma) == "number" && (typeof mb) == "object")
{
    version = "Office 2007"
}
```

A measurement dimension is added by using ActiveXObject, which increase the success rate for tracing visitors. Also, ActiveXObject instances are more stable as secondary attributes than *navigator* objects in browsers.

B. Browser storage mechanisms

Our enhanced solution employs browser storage mechanisms to keep tracing visitors, on account of the usage of spoofing extensions or dynamic reconfiguration tools (i.e. Blink) which lead to changes of fingerprints. Another advantage of using this technique is that we can trace visitors even though they use multiple browsers.

1) Standard HTTP Cookies: From our point of view, browser fingerprinting and the traditional tracing method based on cookies are not opposite, but complementary. Actually, it is fragile for tracing visitors to use fingerprint only. When a browser executes our fingerprinting scripts, they will set up a cookie to log the generated fingerprint at local host, and when the browser visits the website again, the fingerprint logged in the cookie will be sent to the server together. Even though visitors want to hide themselves by using techniques discussed before, resulting in changes of fingerprints, the cookies will provide powerful references to determine what previous fingerprints they were recorded.

2) Local Shared Objects: Local shared objects (LSOs), also known as Flash cookies, are pieces of data that websites which use Adobe Flash store on a visitor's computer. Local shared objects contain data stored in the Action Message Format by individual websites. By default, a SWF application running in Flash Player may store up to 100KB of data to the visitor's hard drive, and LSO data is shared across browsers on the same device. For example, a visitor accesses a website using Firefox browser and views a page containing a video, then closes the Firefox browser, the information about the video can be stored in the LSOs. And then the visitor opens the same page viewed before using Internet Explorer on the same device, the website can read the LSOs values in the Internet Explorer. In this way, we can conduct cross-browser fingerprinting. According to *evercookie*, other browser storage mechanisms (e.g., Silverlight Isolated Storage, the Java JNLP Persistence Service or the Java CVE-2013-0422 exploit cookie) can achieve similar functions.

3) HTML5 Local Storage: HTML5 local storage[9], also called Web storage, is a persistent key-value store in the browser. It is similar to cookies with a full JavaScript API to set or get values but with a greatly enhanced capacity. For HTML5 local storage, the information is not stored in the HTTP request header but in the *localStorage* object and isn't sent to the server as it is with cookies. In our solution, we adopt this storage mechanism to keep persistence, especially for the situation of cookie disabled.

C. Correlation Algorithm

The core of our approach for tracing visitors is that we design a correlation algorithm to compare different fingerprints our background system collects, aiming at finding out fingerprints from a single platform. In this part, we will illustrate the detailed realization of the correlation algorithm we designed and the defined similarity metric which is used to compare different fingerprints.

1) Determining the weight of each attribute of a fingerprint: To calculate the correlation of two different fingerprints, each attribute of the fingerprint as a measurement should be assigned a weight. First, considering three factors (e.g., importance, difficulty of changing, frequency of changing), we grade each attribute of the fingerprint to evaluate, as seen in TABLE II. The factor 'importance' is selected for the reason that an item may have influence on other items, such as some critical items; the 'difficulty of changing' shows how difficult it is to change a item's value, the easier it is, the higher score it gets; the 'frequency of changing' represents the frequency that visitors change the attributes of their fingerprints, we take the EFF's (Electronic Frontier Foundation) findings as the basis of the value to make our paper more rigorous. And then we made a chart to score the attribute values which numbered between 0 ~ 3 (including boundary) is enough to evaluate all the items. Through scoring by experts, we got the chart and figured out a final number of each item. Then, based on the scores, we use Precedence Chart (if more important graded 1, less important graded 0, equally important graded 0.5) to calculate the weight of each attribute, as shown in TABLE III.

TABLE II.

SCORE EACH ITEM BASED ON THREE FACTORS

item	importance	difficulty of changing	frequency of changing	result
userAgent	3	3	3	9
language	1	3	3	7
color_depth	1	1	0	2
resolution	2	3	2	7
available_resolution	2	3	2	7
timezone_offset	1	3	2	6
session_storage	1	0	0	1
local_storage	1	0	0	1
indexed_db	1	0	0	1
add_behavior	1	0	0	1
open_database	1	0	0	1
cpu_class	1	0	0	1
navigator_platform	1	2	1	4
do_not_track	1	3	3	7
regular_plugins	1	2	2	5
canvas	1	0	0	1
webgl	1	0	0	1
adblock	1	0	0	1
hasLiedLanguages	1	0	0	1
hasLiedResolution	1	0	0	1
hasLiedOs	1	0	0	1
hasLiedBrowser	1	0	0	1
touchSupport	1	0	0	1
js_fonts	1	0	0	1

TABLE III

PRECEDENCE CHART

Attribute	user_agent	language	color_depth	···	has_lied_browser	touchSupport	js_fonts	Aik	weight(n=24)
user_agent	0.5	1	1	···	1	1	1	23.5	0.081597222
language	0	0.5	1	···	1	1	1	21	0.072916667
color_depth	0	0	0.5	···	1	1	1	15.5	0.053819444
resolution	0	0.5	1	···	1	1	1	21	0.072916667
available_resolution	0	0.5	1	···	1	1	1	21	0.072916667
timezone_offset	0	0	1	···	1	1	1	18.5	0.064236111
session_storage	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
local_storage	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
indexed_db	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
add_behavior	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
open_database	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
cpu_class	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
navigator_platform	0	0	1	···	1	1	1	16.5	0.057291667
do_not_track	0	0.5	1	···	1	1	1	21	0.072916667
regular_plugins	0	0	1	···	1	1	1	17.5	0.060763889
canvas	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
webgl	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
adblock	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
hasLiedLanguages	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
hasLiedResolution	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
hasLiedOs	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
hasLiedBrowser	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
touchSupport	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667
js_fonts	0	0	0	···	0.5	0.5	0.5	7.5	0.026041667

2) Preprocessing the values of the attributes: Before calculate the correlation of two fingerprints, we need to preprocess the values of the attributes of fingerprint samples. We take three kinds of attributes into consideration. First, the length of the values of attributes is short, for example, the *language* object is usually zh-CN or en-US and the *resolution* object is usually 1280*800 or 1920*1080, not more than 10 letters. We make this kind of values remain unchanged and mark this kind of attributes as $attr_s$; Secondly, the length of the values of attributes is medium and it is inappropriate to directly determine the similarity according to whether two string values are the same or not. For example, the *User-Agent* of Firefox is usually like this: *Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/20100101 Firefox/20.0*, and if the user updates the Firefox, the *User-Agent* may change to this: *Mozilla/5.0 (Windows NT 6.1; WOW64; rv:23.0) Gecko/20130428 Firefox/23.0*. So we cannot arbitrarily determine that they are from different users. We introduce Levenshtein distance algorithm[10] to compute similarity of these attributes which are marked as $attr_m$. If the result is smaller than the threshold we set, we will judge that these two values of attributes are from a single platform; Thirdly, the length of the values of attributes is long and it is unsuitable to use Levenshtein distance algorithm to compare because the threshold is hard to set up, for example, the plugins and the Canvas. In this case, we adopt SimHash[11][12] algorithm to compare these long string attributes which are marked as $attr_l$. We also set up a threshold for the Hamming distance[13] of the results of SimHash of two attributes to make final decision.

3) Similarity metric: To trace visitors based on the correlation of fingerprints, we need to quantify the similarity between fingerprints. We define the following similarity metric aiming at determining if two different fingerprints can be related to a single platform:

$$S(FP_1, FP_2) = SimHash(\{Equal(attr_s(FP_1), attr_s(FP_2)) \cup LevenshteinDistance(attr_m(FP_1), attr_m(FP_2)) \cup SimHash(attr_l(FP_1), attr_l(FP_2))\}, \{W_i\})$$

To compare two browser fingerprint samples, we first preprocess each attribute of the fingerprint. We will not change the values of $attr_s$ attributes; Then we calculate the Levenshtein distances of the $attr_m$ attributes of two fingerprints, and if the result is smaller than the threshold, these two attributes will be both changed to the value of *true*, if not, one of the attribute will be changed to the value *true* while another will be changed to the value of *false*. For $attr_l$ attributes, it is similar to $attr_m$ attributes, just changing the algorithm from Levenshtein to SimHash. Finally, we use current values of attributes as features and the weight computed before to run the SimHash again. The Hamming distance of final SimHash results of two fingerprints is the similarity metric we defined.

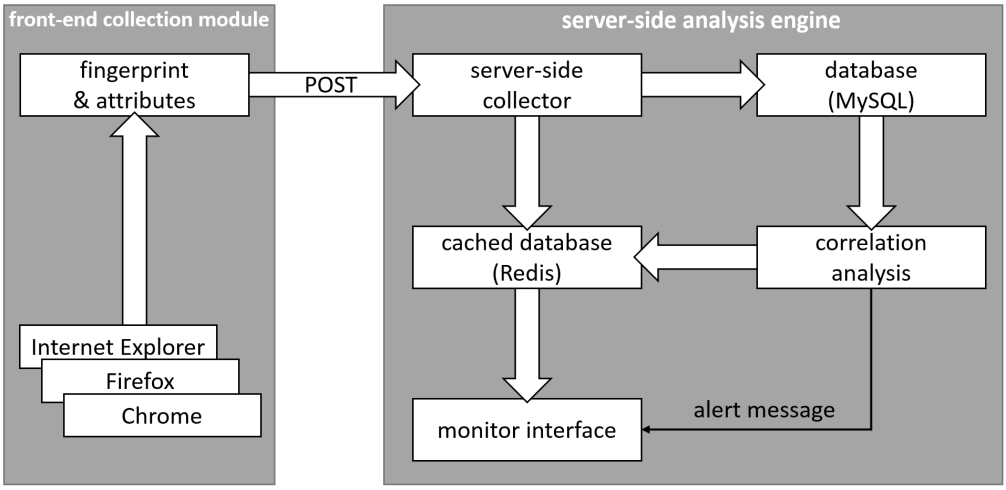

Fig. 2. The architecture of our system

IV. IMPLEMENTATION

This section describes the architecture of the system we developed to trace visitors and how we deploy it. We also take cross-browser of the front-end collection scripts into consideration, because parts of the attributes are just available for some browsers, but not for all. For example, ActiveXObject can be only used in Internet Explorer, while getting intranet IP by using WebRTC leak is only available in Firefox and Chrome. In the last part of this section, we discuss the improvement of future work.

A. Architecture and Deployment

To experiment the effectiveness of our solution to trace visitors continuously, we deploy the collection and analysis system. The system includes two main parts, the front-end collection module and the server-side analysis engine. The scripts of the front-end collection module run when a user (or an attacker) visits our websites, and they collect the attributes and calculate the fingerprints in the browsers. Then, the scripts POST the fingerprints with the corresponding attributes back to our server-side analysis engine by creating hidden form elements. The server-side analysis engine stores the fingerprints and the collected attributes into the database (e.g., MySQL) and create a copy to store in the cached database (e.g., Redis) to be displayed on the monitor interface. The correlation algorithm calculates the similarity of fingerprints from the database uninterrupted in bypass mode. Once the analysis engine find out two different fingerprints are from a single platform, it will push them into the buffer queue in the Redis and send a message of alert to the monitor interface. The architecture of the system is shown in Fig. 2.

We deploy the system in a controlled experimental environment. Fingerprint collection scripts are embed in the website we build. The server-side analysis engine is also deployed in the same network segment which receives data from other intranet hosts. 20 volunteers help us generate fingerprint samples by randomly changing their browser fingerprints via blocking extensions or spoofing extensions. After three months, the items in our background database increased more than 4000, which is enough for us to conduct correlation analysis.

Considering privacy issues, we didn't deploy this system on the public Internet to collect real fingerprints. However, in the near future, when we improve the privacy policy, we will deploy it in the public web with our monitor interface. The monitor interface can present each visit by drawing a line from the source of a visitor to the target website we embed scripts in on the world map. Detailed information (e.g., time stamp, fingerprint, public IP, intranet IP, platform, position) will be also displayed on the interface. By using early warning mechanism and visualization of fingerprint collecting, it is convenient to monitor attack events. And we can get rid of maddening traffic logs[14][15] and access logs[16] every day.

B. Cross-browser Issue

As is known to all, the support of JavaScript varies among different browsers and the syntax of accessing a same object can be different as well. Also, parts of attributes we want to collect may not be available in some browsers. So, if we expect the front-end collection scripts to be robust among browsers, we need to increase the compatibility of the JavaScript code. The idea of the solution is that we import JS scripts conditionally based on the browser types. For example, we will not import the JS scripts which contain ActiveXObject into Firefox or Chrome, neither will we import scripts of WebRTC leak into Internet Explorer. The nuances of different browser versions also trouble us in code-compatible test, since access methods for a small parts of BOM objects are not exactly the same, especially in lower versions.

The server-side programs need to conduct data pre-processing, since parts of the attributes from different browsers may be missing. We give each missing attribute a default value in order to run correlation algorithm successfully. Another consideration is to ignore the access data of Web crawlers whose fingerprints are meaningless for us but cost storage space.

TABLE IV.

CALCULATE THE AVERAGE HAMMING DISTANCE OF SIMHASH

item	change	Hamming distance
user_agent	change UA information	7.0
language	en-us/en/zh	2.0
resolution	1280*800/1152*864	3.0
available_resolution	(change with resolution)	3.0
timezone_offset	(UTC+03:30)/(UTC+12:00)	3.0
navigator_platform	Mac/Linux	3.0
do_not_track	1	4.0
regular_plugins	disable several plugins	5.0
webgl	disabled	3.0
	average	**3.7**

C. *Discussing the Improvement of Future Work*

By now, our solution of enhanced browser fingerprinting increases the accuracy of tracing web visitors. For little change of the attribute leading to completely change of the fingerprint, our system can relate these two fingerprints and determine they are from the same origin, which makes it possible for tracing in a single website.

As is known to all, browser fingerprinting has the ability of cross-origin as it is with cookies[17]. If two independent websites deploy our fingerprint collecting scripts, fingerprints from both of these two sites will be sent back to our analysis engine. The values of these fingerprints are uncorrelated to the websites, just related to the browsers. So, in future work, we want to conduct cross-site tracing which allows us to associate a visitor with another one who visits the other website.

V. RESULTS

We have deployed a web blog in the intranet with fingerprinting scripts in the home page to collect fingerprints from different intranet hosts. In the past three months, we have collected more than 4000 fingerprints from over 20 hosts we tested on.

We calculate the average of the Levenshtein distance when changing parts of the *User-Agent*, and the result is 19.96. So we set up 20 as the threshold of the Levenshtein distance of *User-Agent*. For the fonts, we set up 10 as the threshold depending on the average length of the names of the fonts. To calculate the threshold of the SimHash algorithm of plugins, we disable each plugin in the browser to compute the average Hamming distance, which is 6. Finally, we set up 3.7 as the threshold of the similarity metric basing on the average Hamming distance of SimHash algorithm when changing any of the attributes (TABLE IV.).

According the threshold we defined, we determine the correlation of each two fingerprints. We examine 4176 fingerprint samples in the database, and 2335 fingerprints can be related to another collected fingerprint by using the solution we propose, while the number is only 1876 by using traditional fingerprinting technique. The accuracy of tracing visitors increases by 24.5%. For the increment of 459 fingerprints, 273 of which are related because of the usage of browser storage mechanisms, 76 of which are related owing to the secondary attributes we adopt, and the last 110 of which are related by the results of the correlation algorithm. Fig. 3 shows that as follows.

VI. CONCLUSION

This work enhances the ability to trace Web visitors, even Web attackers, than traditional browser fingerprinting techniques. We creatively introduce secondary attributes which are not easy to be changed to provide basis for judging fingerprints. Another improvement by employing browser storage mechanisms allows us to recover visitors' previous fingerprints even though they use blocking extensions or spoofing extensions to change their fingerprints. We also contribute a fingerprint correlation algorithm and a similarity metric to evaluate the relation between two fingerprints. We deploy the collection and analysis system we developed to experiment the effectiveness of our solution. We compute the average of the Levenshtein distance and the Hamming distance for changing attributes to set up a threshold to determine whether two different fingerprints are from a simple origin.

This work is helpful for tracing Web attackers since they always hide themselves by using VPN or anonymous networks. Our collection and analysis system can be used to conduct correlation analyzing to realize continuous tracing of the attackers. In the future work, we plan to increase the capacity of our system by conducting cross-site tracing.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers f or their helpful comments for improving this paper. This work i s supported by the National Natural Science Foundation of Chi na under grant (No. 61303239) and the National High Technol ogy Research and Development Program (863 Program) of Chi na under grant (No. 2012AA012902).

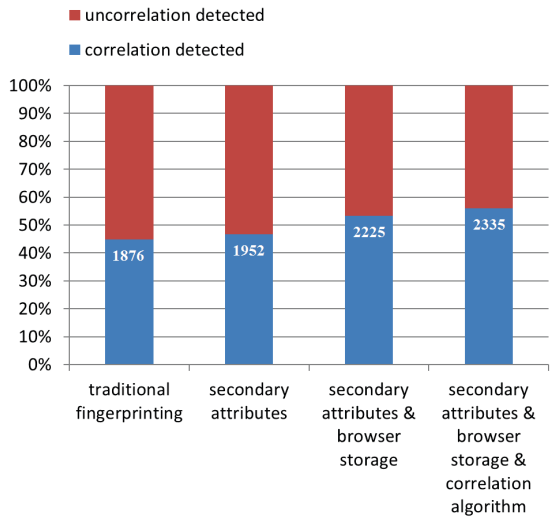

Fig. 3. Comparison of traditional fingerprinting and our enhanced solution

REFERENCES

[1] J. S. Park and R. Sandhu, "Secure cookies on the Web," in *IEEE internet computing*, 2000, 4(4): 36-44.

[2] V. Pimentel and B. G. Nickerson, "Communicating and displaying real-time data with WebSocket," in *Internet Computing, IEEE*, 2012, 16(4): 45-53.

[3] P. Eckersley, "How unique is your web browser?" in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, ser. PETS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1-18.

[4] Pierre Laperdrix, Walter Rudametkin and Benoit Baudry, "Mitigating Browser Fingerprint Tracking: Multilevel Reconfiguration and Diversification," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2015 IEEE/ACM 10th International Symposium on*, Florence, 2015, pp. 98-108.

[5] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in *Proceedings of W2SP 2012*, M. Fredrikson, Ed. IEEE Computer Society, May 2012.

[6] A. Chaabane, P. Manils and M. A. Kaafar, "Digging into anonymous traffic: A deep analysis of the tor anonymizing network," in *Network and System Security (NSS), 2010 4th International Conference on. IEEE*, 2010, pp. 167-174.

[7] Gaofeng He, Ming Yang, Xiaodan Gu, Junzhou Luo and Yuanyuan Ma, "A novel active website fingerprinting attack against Tor anonymous system," in *Computer Supported Cooperative Work in Design (CSCWD), Proceedings of the 2014 IEEE 18th International Conference on. IEEE*, 2014, pp. 112-117.

[8] M. Adeyeye, I. Makitla and T. Fogwill, "Determining the signalling overhead of two common WebRTC methods: JSON via XMLHttpRequest and SIP over WebSocket," in *AFRICON, 2013. IEEE*, 2013, pp. 1-5.

[9] S. Z. Naseem and F. Majeed, "Extending HTML5 local storage to save more data; efficiently and in more structured way," in *Digital Information Management (ICDIM), 2013 Eighth International Conference on. IEEE*, 2013, pp. 337-340.

[10] S. Rane and W. Sun, "Privacy preserving string comparisons based on Levenshtein distance," in *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on. IEEE*, 2010, pp. 1-6.

[11] Manku G S, Jain A, Das Sarma A, "Detecting near-duplicates for web crawling" in *Proceedings of the 16th international conference on World Wide Web. ACM*, 2007, pp. 141-150.

[12] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. 34th Annual Symposium on Theory of Computing (STOC 2002)*, pages 380-388, 2002.

[13] G. S. Shehu, A. M. Ashir and A. Eleyan, "Character recognition using correlation & hamming distance" in *Signal Processing and Communications Applications Conference (SIU), 2015 23th. IEEE*, 2015, pp. 755-758.

[14] K. Yoshioka and T. Matsumoto, "Fingerprinting traffic log," in *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IIHMSP'08 International Conference on. IEEE*, 2008, pp. 143-146.

[15] Y. Yang, C. Huang and Z. Qin, "A Network Misuse Detection Mechanism Based on Traffic Log," in *Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC'09. International Conference on. IEEE*, 2009, 1: 526-529.

[16] C. H. Lin, J. C. Liu and C. R. Chen, "Access log generator for analyzing malicious website browsing behaviors," in *Information Assurance and Security, 2009. IAS'09. Fifth International Conference on. IEEE*, 2009, 2: 126-129.

[17] M. Dhawan and V. Ganapathy, "Analyzing information flow in JavaScript-based browser extensions" in *Computer Security Applications Conference, 2009. ACSAC'09. Annual. IEEE*, 2009, pp. 382-391.