Combatting Browser Fingerprinting with ChromeDust

Ram Bhaskar <rambhask@mit.edu>
Rishikesh Tirumala <rrt@mit.edu>
Timmy Galvin <tgalvin@mit.edu>
6.858 Final Project (Lab 7)
December 12, 2013

Introduction

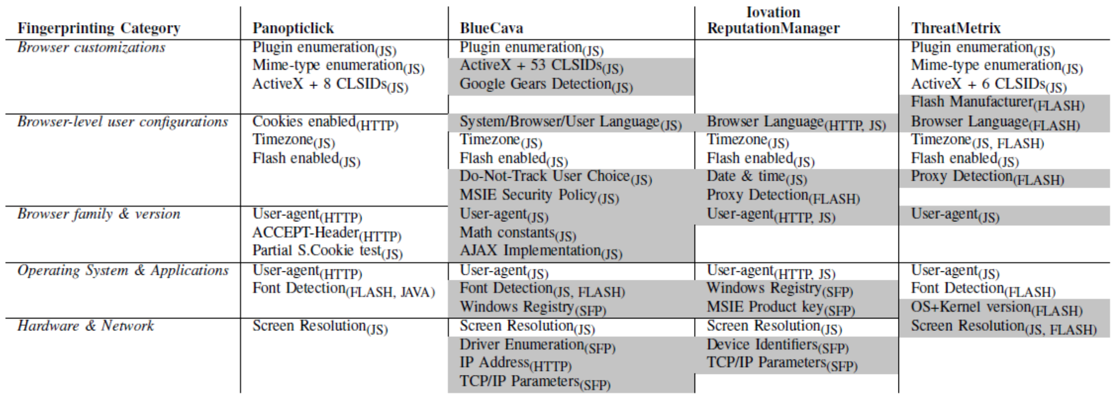
Online advertising is a multi-billion dollar industry. There exists a constant pressure to improve the techniques used to further influence consumer habits. At the core of online advertising efforts lies directed advertisements–ads shown for a specific user to align with their interests. This individualized approach is mainly accomplished by tracking a user's online browsing habits to build a model of his/her interests and behaviors. The main way to track a user's browsing activity since 1994 has been cookies. However, cookies have recently faced both legal and technical limitations, so advertisers have adopted a new method to track user activity: browser fingerprinting. Fingerprinting is the process of identifying a user using various metadata accessible through their browsers (e.g., user agent string, screen resolution). This method has gained prominence in the last four years and stands as a threat to users' privacy.

Problem Statement

While users have methods with which to protect themselves from tracking cookie-based tracking, attempts to interfere with browser fingerprinting are next to non-existent. The reason for this void of options is multi-fold: the technology, more specifically the approach, only became well-known in 2009, solutions must be constantly updated, and the attack surface is not well-defined. While these are daunting challenges, we believe the solution is an open-source browser extension that interferes with common fingerprinting techniques without sacrificing usability too much. So while our research made it clear that we could not stop fingerprinting, we hope our project will help raise awareness of browser fingerprinting and focus efforts on generating solutions.

A Closer Look at Fingerprinting

Although it may not seem like it, fighting fingerprinting is a complicated task. There are so many parameters that can act as personal identifiable information (PII) and it is near impossible to protect against all of these. Different levels of fingerprint tracking are implemented in today's web ecosystem. The first is that of a source that collects information from HTTP Headers and JavaScript objects, and stores these values. Then, once a repeat fingerprint is observed, adversaries can match the data and begin to collect information and track users' actions. The second kind is much more sophisticated. These are mostly commercial fingerprinting services such as BlueCava, Iovation, and ThreatMatrix. These services go above and beyond the rudimentary tracking mechanisms and extract additional information from the user's browser. Figure 1 shows a comparison between Panopticlick and these other three companies in terms of what features they are tracking in their applications.



Fingerprinting Category	Panopticlick	BlueCava	Iovation ReputationManager	ThreatMetrix
Browser customizations	Plugin enumeration$_{\text{(JS)}}$ Mime-type enumeration$_{\text{(JS)}}$ ActiveX + 8 CLSIDs$_{\text{(JS)}}$	Plugin enumeration$_{\text{(JS)}}$ ActiveX + 53 CLSIDs$_{\text{(JS)}}$ Google Gears Detection$_{\text{(JS)}}$		Plugin enumeration$_{\text{(JS)}}$ Mime-type enumeration$_{\text{(JS)}}$ ActiveX + 6 CLSIDs$_{\text{(JS)}}$ Flash Manufacturer$_{\text{(FLASH)}}$
Browser-level user configurations	Cookies enabled$_{\text{(HTTP)}}$ Timezone$_{\text{(JS)}}$ Flash enabled$_{\text{(JS)}}$	System/Browser/User Language$_{\text{(JS)}}$ Timezone$_{\text{(JS)}}$ Flash enabled$_{\text{(JS)}}$ Do-Not-Track User Choice$_{\text{(JS)}}$ MSIE Security Policy$_{\text{(JS)}}$	Browser Language$_{\text{(HTTP, JS)}}$ Timezone$_{\text{(JS)}}$ Flash enabled$_{\text{(JS)}}$ Date & time$_{\text{(JS)}}$ Proxy Detection$_{\text{(FLASH)}}$	Browser Language$_{\text{(FLASH)}}$ Timezone$_{\text{(JS, FLASH)}}$ Flash enabled$_{\text{(JS)}}$ Proxy Detection$_{\text{(FLASH)}}$
Browser family & version	User-agent$_{\text{(HTTP)}}$ ACCEPT-Header$_{\text{(HTTP)}}$ Partial S.Cookie test$_{\text{(JS)}}$	User-agent$_{\text{(JS)}}$ Math constants$_{\text{(JS)}}$ AJAX Implementation$_{\text{(JS)}}$	User-agent$_{\text{(HTTP, JS)}}$	User-agent$_{\text{(JS)}}$
Operating System & Applications	User-agent$_{\text{(HTTP)}}$ Font Detection$_{\text{(FLASH, JAVA)}}$	User-agent$_{\text{(JS)}}$ Font Detection$_{\text{(JS, FLASH)}}$ Windows Registry$_{\text{(SFP)}}$	User-agent$_{\text{(HTTP, JS)}}$ Windows Registry$_{\text{(SFP)}}$ MSIE Product key$_{\text{(SFP)}}$	User-agent$_{\text{(JS)}}$ Font Detection$_{\text{(FLASH)}}$ OS+Kernel version$_{\text{(FLASH)}}$
Hardware & Network	Screen Resolution$_{\text{(JS)}}$	Screen Resolution$_{\text{(JS)}}$ Driver Enumeration$_{\text{(SFP)}}$ IP Address$_{\text{(HTTP)}}$ TCP/IP Parameters$_{\text{(SFP)}}$	Screen Resolution$_{\text{(JS)}}$ Device Identifiers$_{\text{(SFP)}}$ TCP/IP Parameters$_{\text{(SFP)}}$	Screen Resolution$_{\text{(JS, FLASH)}}$

Figure 1: List of features used by Panopticlick and other fingerprinting services.[1]

As one may notice, these companies track much more sophisticated information than their basic counterparts. One common exploit is through Flash. There are specific vulnerabilities

[1] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting. In IEEE Symposium on Security and Privacy, San Francisco, CA, USA, May 2013.

that allow for font detection, proxy detection, and even Windows Registry and IP information which could prove to be enough to identify an user uniquely. Many other tricks involve the navigator and the screen object of the window. These objects, the navigator in particular, are very powerful as they contain incredible amount of information. For example, the navigator has properties such as *userAgent*, *plugins*, *appName*, *platform*, *language*, etc. However, there are also properties of the navigator object that are unique to certain browsers. For example, Firefox has properties such as *mozAlarms*, *mozDoNotTrack*, *mozSettings*, etc that are usually only prevalent in Firefox. Similarly, Internet Explorer has properties such as *msPointerEnabled*, *msMaxTouchPoints*, etc. that only exist in IE. Adversaries can use the information present to detect which browser users are using. Also, different versions of the same browser have different properties as well, which also complicates protection against this detection scheme. Other tricky ways of fingerprinting utilized by these services include math constant detection, using css size to determine fonts, assessing the mutability of navigator object, preserving enumeration of JavaScript objects, and detecting mismatch in userAgent information gather from http headers and JavaScript objects.

Code Design and Implementation

Our code follows the basic structure of a Chrome Extension. On a high level, the *manifest.json* file holds information about the extension as a whole, including what permissions are required from the user. We ask for access to local storage, tabs, and the http headers that the user's browser sends to various websites. The manifest also identifies the various javascript files where most of the extension's functionality lies.

The user interfaces with the extension through *options.html* and *options.js*, the options page which is opened through the extension's menu. The options page allows a user to see which pieces of PII are available to manipulate. Previously, we had allowed users to choose what to spoof their data as, but we removed that functionality. Our scheme works best when each user is seen as unique over the same sites and sessions, since the randomness prevents a site from being able to track a user over his/her trawl across the world wide web. The options page handles the storing and displaying of this data to the user, along with passing messages to *background.js*, which holds the state machines relating to user preferences.

Inside *background.js*, each potential PII data-point is given a boolean variable, mapping to whether or not the user wants to spoof that data-point. *Background.js* handles listening for messages from other areas of the extension (specifically *options.js*), along with user-triggered actions that we are interested in. When the user switches tabs, we are able to spoof his/her http headers, specifically the user-agent string. This file also handles sending variable states to *content.js*, which handles client-side JavaScript.

One of the roadblocks we ran into while developing ChromeDust was the realization that Chrome extensions execute in a different executable environment from the web page client-side JavaScript code. By changing code in the extension's JavaScript executable environment, we were not able to spoof what the server-side code could see. Instead, we inject JavaScript code into the DOM to spoof PII attributes on to the page that the user is viewing, and this functionality is handled in *content.js*. The file also holds an array of plausible user-agent strings and version numbers so that there are a large number of permutations of valid configurations.

Conclusion

We believe that our project is the beginning of an open-source solution to browser fingerprinting in the form of a browser extension. We focused mainly on interfering with common fingerprinting techniques while maintaining usability as best we could. We hope that others will build upon the work that we have done so far and will continue to raise awareness of browser fingerprinting while providing users with an easy-to-use solution.