Fingerprinting Web Users Through OpenType Font Features

Matthew Cameron
Cowboy Lynk
Aramis Subee
Lili Sun

***Abstract*— This paper covers our team's final project for 6.857 Spring 2020. We attempt to fingerprint users with only variations between browser font features. There's a surprising amount of variance in these feature settings between operating systems and web browsers. A past paper by Fifield and Egelman looked into similar fingerprinting but only altered simple CSS features such as serif versus sans-serif fonts. In this report, we use OpenType features instead. OpenType provides a much richer feature set and better crossplatform compatibility to gain information from all types of users.**

I. Introduction

A. Fingerprinting

Fingerprinting attempts to identify a user on the web with their browser and device settings. Its main use is in advertising, where companies attempt to profile users to better cater ads to them. Fingerprinting is also used by researchers to identify botnets because of the systems' similarity.

There are a wide array of approaches to fingerprinting. Some past examples have used HTTP request headers, browser plugins, and time zone.

The most recent advancements in fingerprinting, called Canvas Fingerprinting, comes with the introduction of HTML5. Whereas in the past fingerprinting required loading something onto a user's computer, Canvas Fingerprinting only examines the values within HTML5 canvas elements. This means most client-side protections against malware are ineffective, because all the data is collected on the server. Our paper is similar to this category of fingerprinting. We measure text rather than shapes but still maintain server-side control of the data collection.

II. Previous Work

A. Fingerprinting with Fonts

Our work draws mostly from a paper by Fifield and Egelman. They too attempt to use bounding boxes of Unicode glyphs to fingerprint their users. However, they create variance by applying simple CSS features to each glyph. For example, they test the difference between serif versus sans-serif fonts. With this technique, they were able to uniquely identify 34% of their volunteers. We extend their work by using OpenType rather than CSS.

B. Measuring Entropy

A paper by Eckerley famously was able to identify 84% of its users with a browser's version and configuration being sent to their website, Panopticlick. The experiment has been running since 2010 and is still ongoing. The point of interest in this study is how they used entropy as a measure of distinction between feature settings. We follow suit, and also employ entropy as our method of measurement.

III. OpenType

OpenType is a font file format developed by Adobe and Microsoft. It has been widely adopted, and now every mainstream OS and browser has some support for it. This allows us to gain information from users no matter what machine or browser they're running. Some fingerprinting methods fail due to browser dependencies, and we escape that pitfall. Also, OpenType has a vast feature set. It includes support for kerning, ligatures, specific language support, and many other features. In theory, we should be able to gain more information

on our users than Fifield and Egelman due to our larger feature set.

IV. Data Collection

A. Methodology

To build a fingerprint, we need to identify differences between users based on observable differences in browser, OS, installed fonts, software versions, configurations, and other parts of their system. We first needed to collect data on how OpenType features lead to measurable differences between different users.

Building off of the previous work by Fifield and Egelman, we decided to measure the width and height of a variety of glyphs with different OpenType features applied. This is done in the hopes that we will reveal differences in sizes based on different default fonts and browser support and implementation, which can lead to measurable differences for distinct users.

To achieve this, we do the following: First, we apply every setting for every OpenType feature individually. Then, for every setting, we draw each of our chosen glyphs, one at a time. For every glyph, we measure and store the width and height of the glyph. Finally, we save the results for each user so we can later analyse how well we can fingerprint users.

B. Implementation

In this section we will discuss our specific implementation and some challenges we faced. To collect our data, we built a website that would perform measurements for a user and insert the data into a database on our server for evaluation. Our website uses simple HTML and CSS to show instructions and create a button that allows volunteers who visit our site to submit our data. The HTML and CSS also create a hidden element that allows us to draw and measure glyphs without being shown to the user. The bulk of our website's functionality comes from our JavaScript.

When a visitor clicks on the button to generate and collect their data, a JavaScript function is called which does the following:

1) Check for a specific cookie to be set
   - If the cookie is set, then this user has already been fingerprinted and we stop here
   - If the cookie is not set, we assume the user has not been fingerprinted and continue
2) Loop through the CSS built-in OpenType font variants and the built-in settings for each
   a) Apply this setting for this feature
   b) Loop through our chosen set of characters
      i) Draw the gylph
      ii) Measure the glyph and add the measurements to a dictionary
   c) Add the dictionary of glyph measurements to a dictionary mapping settings to measurements
3) Loop through the CSS OpenType font feature settings (more on this difference later)
   a) Apply this setting
   b) Loop through our chosen set of characters
      i) Draw the gylph
      ii) Measure the glyph and add the measurements to a dictionary
   c) Add the dictionary of glyph measurements to a dictionary mapping settings to measurements
4) Send these JSON encoded dictionaries along with the user-agent string to the server
   - If the data transfer is successful, then set a cookie to prevent duplicate entries, and display a success message to the user

When implementing our website, we encountered a few challenges. One such challenge was the difference between the CSS built-in OpenType font variants (e.g. `font-variant-caps: all-small-caps`), and the CSS OpenType font features (e.g. `font-feature-settings: "smcp"`). The built-in settings cover a smaller range of features and have less browser support, but they also sometimes cover combinations of individual features or have settings that cannot be represented by the font features. We chose to do both in order to cover every setting, and it also allows us, in theory, to check if a user's browser supports the font variant as an extra fingerprinting method.

Another issue we had was getting an accurate height measurement. For most characters, the

height measurement depends on the line height, and not the height of the drawn glyph. Improving this measurement could potentially improve our fingerprinting.

Finally, since we are testing over 280 individual settings for each character, and there are over a hundred-thousand Unicode characters, we could not test all of them in a reasonable amount of time. We chose the our characters because they were a small subset that either worked well in prior fingerprinting by Fifield and Egelman or we thought specific OpenType features would apply to them (some are actually multiple characters because certain settings, such as ligatures, only apply to certain combinations of characters). The characters we chose are: `["a", "b", "c", "d", "e", "f", "\u20B9", "\u2581", "\u20BA", "\uA73D", "\uFFFD", "ff", "fi", "0", "1", "2", "3", "4", "麹 町", "しんかんせん", "大学"]`

V. Data Evaluation

A. Fingerprinting Effectiveness

To determine the effectiveness of this method in fingerprinting, we split up all the submissions into equivalence classes. To do this, we look at the measurements of all the glyph-feature combinations for a particular submission and see if they are all equal with those of another submission.

Out of our 118 collected submissions, we had 48 equivalence classes. Out of these 48, 34 included just one submission, indicating that 28.8% of the users had a unique fingerprint just from font metrics (see Figure 1 for size distribution). Additionally, the largest equivalence class, with 30 submissions, all proved to have come from an iOS device with version 13 of software. Every equivalence class that included more than one submission had user agent strings almost all equivalent, with sometimes just one or two deviating in a small detail, not the OS or browser.

This shows that with our fingerprinting technique from our set of glyph-feature combinations, we can identify a user's OS and browser, and sometimes additional device settings.

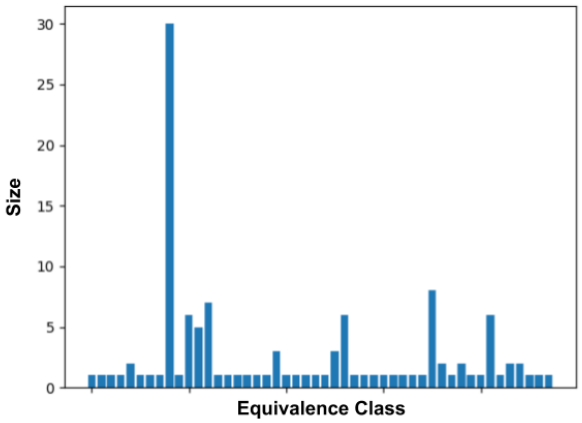

Fig. 1. Size distribution of equivalence classes.

B. Information Value of Glyph-Feature Combinations

To determine how much information a particular glyph-feature combination gave us, we calculated the entropy. We do it like so:

$$H(S) = -\sum_{v \in S} P_s(v) \cdot \log_2 P_s(v).$$

In the formula above, S corresponds to some glyph-feature combination. The value v is some specific dimension measurement of S within the submissions. $P_s(v)$ is the number of times v shows up as the measurement of S across all submissions divided by the total number of submissions.

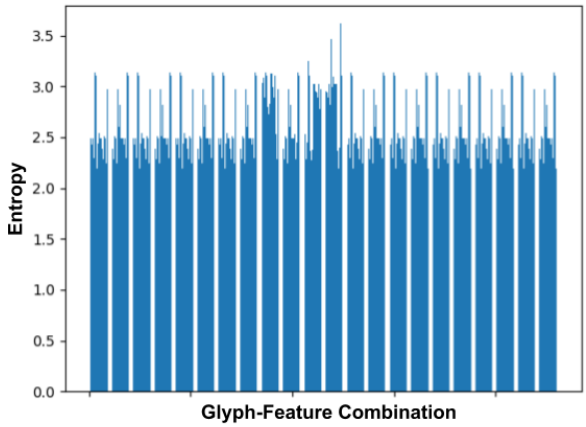

Fig. 2. Entropy of feature-glyph combinations.

Our data shows that the character '\u2581' with the feature font-variant-caps unicase applied to

it has the largest entropy of 3.616 (indicated by the peak in Figure 2). This was followed by the character '\uFFFD' with the feature font-variant-caps all-small-caps applied to it, with an entropy of 3.466. We notice from looking at the graph that there is a slight repeated pattern in sequence of entropies. This tells us that most features did not change the entropies by a large amount, just the features displaying irregular blocks of entropies around the middle.

For reference, we can see roughly how the entropy of glyph-feature combinations compares against entropy of just glyphs shown in Figure 3.

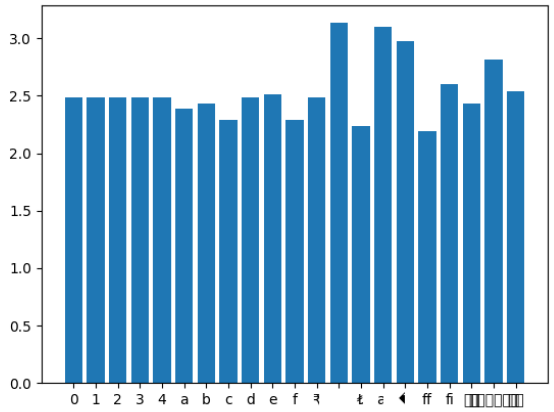

Fig. 3. Entropy of glyphs without OpenType features applied.

C. Information Value of OpenType Features

Another thing we wanted to evaluate was how much extra information OpenType features gave us in comparison to measuring glyphs with no features. Say our set of features is F and set of glyphs is G. Then, we have entropy values $H(S_{fg})$ which denotes the entropy defined above for certain glyph-feature combination where $g \in G$ and $f \in F$. To calculate the information value of a certain OpenType feature $f \in F$, we take the average of the difference of the entropy of glyph-feature combinations with the said fixed f and the entropy of the glyph without features:

$$\frac{1}{|G|} \sum_{g \in G} H(S_{fg}) - H(S_g).$$

Our data shows that the feature that has the greatest average change in entropy is applying the subscript font variant position on glyphs; its average change in entropy is 0.425. Following this is superscript on font variant position, with similar change in entropy. Following this are different capitalization features. See Figure 4 for the average change in entropy across applying different OpenType features to glyphs.

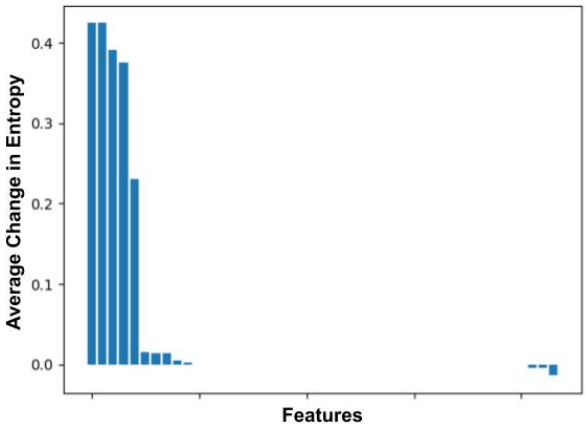

Fig. 4. Average change in entropy for OpenType features.

VI. Future Work

A. Preventing Duplicate Database Entries

One of the more straightforward ways to improve our approach is to better prevent duplicate entries in the server. As we discussed earlier, fingerprinting attempts to identify an individual using information about their machine, gathered through their web browser. However, to properly evaluate the effectiveness of our method we need to ensure that each fingerprint we collect comes from a unique user. This is to say that we need to prevent duplicate entries.

Currently, we approach this problem with a very simple cookie that prevents users from sending results twice. The value of the cookie is a boolean that indicates whether or not the user has already sent data to our server. However, cookies are stored on the client side and can easily be modified or deleted, thus making our approach for detecting duplicate fingerprints not very reliable.

In the future we could require some other form of authentication when collecting data. For example, when sending out the form to MIT students,

we could authenticate them with Kerberos' before accepting their data. Ensuring that each data entry on our server is unique, will lead to a more accurate measure of our approach's entropy.

B. Better Entropy-driven Glyph Selection

As previously mentioned, our web app attempts to fingerprint a user by displaying a small set of 21 glyphs one-by-one in a hidden div and observing their dimensions. These dimensions may vary depending on various characteristics of the user's machine. As such, we are able to take advantage of these differences to identify users. However, this only works if the glyphs we choose vary across machines. Unfortunately, some glyphs are constant or have little variation between machines.

Testing all glyphs would solve this issue because it ensures that we measure all glyphs with variation. However, this approach is not very practical. The reason being that testing all glyphs is computationally expensive and we did not want volunteers sending us data to have to wait up to 10 minutes. This is the reason we went with a 21 glyph set that runs in under 10 seconds. We figured the faster we collected data, the more people that would be willing to send us data. However, we do not know if the limited number of glyphs we selected gives us the most information about the user.

A better solution would be to test all glyphs with a small group of people that are okay with waiting up to 10 minutes. In this way we can experimentally determine which glyphs have the highest entropy. Then we could then use the top 10 glyphs from this small study to send out the site to more people and collect more data without requiring everyone to wait for up to 10 minutes.

C. Defenses Against OpenType Fingerprinting

Finally, if our method proves to be successful, we would like to explore ways to defend against fingerprinting using our method. Fingerprinting is made more difficult, in general, by reducing differences across systems. By shipping a set of standard fonts, thus limiting the variation due to font file availability, font fingerprinting is made much less effective. We are curious to see how our implementation of fingerprinting using features of OpenType holds up against this defense. As we continue to design our extension of the original approach using OpenType features, we will keep in mind how the original approach handled this defense and use this information to inform our own design process.

Additionally, if our approach proves to be successful and resistant to the defense described above, we would like to investigate other defense methods–both feasible and infeasible. In doing so, we hope that we can highlight the strengths and flaws of our method. With this knowledge, we can continue to iterate on the design and make it more effective.

List of Figures

References

[1] David Fifield and Serge Egelman, 2015. "Fingerprinting Web Users Through Font Metrics." International Conference on Financial Cyrptography and Data Security 2015, pp. 107-124.

[2] Peter Eckersley, 2010. "How unique is your web browser?" 10th Privacy Enhancing Technologies Symposium, pp. 1-18.