How Unique Is Your Web Browser?

Peter Eckersley*

Electronic Frontier Foundation,
pde@eff.org

Abstract. We investigate the degree to which modern web browsers are subject to "device fingerprinting" via the version and configuration information that they will transmit to websites upon request. We implemented one possible fingerprinting algorithm, and collected these fingerprints from a large sample of browsers that visited our test side, panopticlick.eff.org. We observe that the distribution of our fingerprint contains at least 18.1 bits of entropy, meaning that if we pick a browser at random, at best we expect that only one in 286,777 other browsers will share its fingerprint. Among browsers that support Flash or Java, the situation is worse, with the average browser carrying at least 18.8 bits of identifying information. 94.2% of browsers with Flash or Java were unique in our sample.

By observing returning visitors, we estimate how rapidly browser fingerprints might change over time. In our sample, fingerprints changed quite rapidly, but even a simple heuristic was usually able to guess when a fingerprint was an "upgraded" version of a previously observed browser's fingerprint, with 99.1% of guesses correct and a false positive rate of only 0.86%.

We discuss what privacy threat browser fingerprinting poses in practice, and what countermeasures may be appropriate to prevent it. There is a tradeoff between protection against fingerprintability and certain kinds of debuggability, which in current browsers is weighted heavily against privacy. Paradoxically, anti-fingerprinting privacy technologies can be self-defeating if they are not used by a sufficient number of people; we show that some privacy measures currently fall victim to this paradox, but others do not.

1 Introduction

It has long been known that many kinds of technological devices possess subtle but measurable variations which allow them to be "fingerprinted". Cameras [1,2], typewriters [3], and quartz crystal clocks [4,5] are among the devices that can be

* Thanks to my colleagues at EFF for their help with many aspects of this project, especially Seth Schoen, Tim Jones, Hugh D'Andrade, Chris Controllini, Stu Matthews, Rebecca Jeschke and Cindy Cohn; to Jered Wierzbicki, John Buckman and Igor Serebryany for MySQL advice; and to Andrew Clausen, Arvind Narayanan and Jonathan Mayer for helpful discussions about the data. Thanks to Chris Soghoian for suggesting backoff as a defence to font enumeration.

entirely or substantially identified by a remote attacker possessing only outputs or communications from the device.

There are several companies that sell products which purport to fingerprint web browsers in some manner [6,7], and there are anecdotal reports that these prints are being used both for analytics and second-layer authentication purposes. But, aside from limited results from one recent experiment [8], there is to our knowledge no information in the public domain to quantify how much of a privacy problem fingerprinting may pose.

In this paper we investigate the real-world effectiveness of browser fingerprinting algorithms. We defined one candidate fingerprinting algorithm, and collected these fingerprints from a sample of 470,161 browsers operated by informed participants who visited the website `https://panopticlick.eff.org`. The details of the algorithm, and our collection methodology, are discussed in Section 3. While our sample of browsers is quite biased, it is likely to be representative of the population of Internet users who pay enough attention to privacy to be aware of the minimal steps, such as limiting cookies or perhaps using proxy servers for sensitive browsing, that are generally agreed to be necessary to avoid having most of one's browsing activities tracked and collated by various parties.

In this sample of privacy-conscious users, 83.6% of the browsers seen had an instantaneously unique fingerprint, and a further 5.3% had an anonymity set of size 2. Among visiting browsers that had either Adobe Flash or a Java Virtual Machine enabled, 94.2% exhibited instantaneously unique fingerprints and a further 4.8% had fingerprints that were seen exactly twice. Only 1.0% of browsers with Flash or Java had anonymity sets larger than two. Overall, we were able to place a lower bound on the fingerprint distribution entropy of 18.1 bits, meaning that if we pick a browser at random, at best only one in 286,777 other browsers will share its fingerprint. Our results are presented in further detail in Section 4.

In our data, fingerprints changed quite rapidly. Among the subset of 8,833 users who accepted cookies and visited `panopticlick.eff.org` several times over a period of more than 24 hours, 37.4% exhibited at least one fingerprint change. This large percentage may in part be attributable to the interactive nature of the site, which immediately reported the uniqueness or otherwise of fingerprints and thereby encouraged users to find ways to alter them, particularly to try to make them less unique. Even if 37.4% is an overestimate, this level of fingerprint instability was at least momentary grounds for privacy optimism.

Unfortunately, we found that a simple algorithm was able to guess and follow many of these fingerprint changes. If asked about all newly appearing fingerprints in the dataset, the algorithm was able to correctly pick a "progenitor" fingerprint in 99.1% of cases, with a false positive rate of only 0.87%. The analysis of changing fingerprints is presented in Section 5.

2 Fingerprints as Threats to Web Privacy

The most common way to track web browsers (by "track" we mean associate the browser's activities at different times and with different websites) is via HTTP cookies, often set by with 3rd party analytics and advertising domains [9].

There is growing awareness among web users that HTTP cookies are a serious threat to privacy, and many people now block, limit or periodically delete them. Awareness of supercookies is lower, but political and PR pressures may eventually force firms like Adobe to make their supercookies comply with the browser's normal HTTP cookie privacy settings.

In the mean time, a user seeking to avoid being followed around the Web must pass three tests. The first is tricky: find appropriate settings that allow sites to use cookies for necessary user interface features, but prevent other less welcome kinds of tracking. The second is harder: learn about all the kinds of supercookies, perhaps including some quite obscure types [10,11], and find ways to disable them. Only a tiny minority of people will pass the first two tests, but those who do will be confronted by a third challenge: fingerprinting.

As a tracking mechanism for use against people who limit cookies, fingerprinting also has the insidious property that it may be much harder for investigators to detect than supercookie methods, since it leaves no persistent evidence of tagging on the user's computer.

2.1 Fingerprints as Global Identifiers

If there is enough entropy in the distribution of a given fingerprinting algorithm to make a recognisable subset of users unique, that fingerprint may essentially be usable as a 'Global Identifier' for those users. Such a global identifier can be thought of as akin to a cookie that cannot be deleted except by a browser configuration change that is large enough to break the fingerprint.

Global identifier fingerprints are a worst case for privacy. But even users who are not globally identified by a particular fingerprint may be vulnerable to more context-specific kinds of tracking by the same fingerprint algorithm, if the print is used in combination with other data.

2.2 Fingerprint + IP address as Cookie Regenerators

Some websites use Adobe's Flash LSO supercookies as a way to 'regenerate' normal cookies that the user has deleted, or more discretely, to link the user's previous cookie ID with a newly assigned cookie ID [12].

Fingerprints may pose a similar 'cookie regeneration' threat, even if those fingerprints are not globally identifying. In particular, a fingerprint that carries no more than 15-20 bits of identifying information will in almost all cases be sufficient to uniquely identify a particular browser, given its IP address, its subnet, or even just its Autonomous System Number.[1] If the user deletes their cookies

[1] One possible exception is that workplaces which synchronize their desktop software installations completely may provide anonymity sets against this type of attack. We

while continuing to use an IP address, subnet or ASN that they have used previously, the cookie-setter could, with high probability, link their new cookie to the old one.

2.3 Fingerprint + IP address in the Absence of Cookies

A final use for fingerprints is as a means of distinguishing machines behind a single IP address, even if those machines block cookies entirely. It is very likely that fingerprinting will work for this purpose in all but a tiny number of cases.

3 Methodology

3.1 A Browser Fingerprinting Algorithm

We implemented a browser fingerprinting algorithm by collecting a number of commonly and less-commonly known characteristics that browsers make available to websites. Some of these can be inferred from the content of simple, static HTTP requests; others were collected by AJAX[2]. We grouped the measurements into eight separate strings, though some of these strings comprise multiple, related details. The fingerprint is essentially the concatenation of these strings. The source of each measurement and is indicated in Table 3.1.

In some cases the informational content of the strings is straightforward, while in others the measurement can capture more subtle facts. For instance, a browser with JavaScript disabled will record default values for `video`, `plugins`, `fonts` and `supercookies`, so the presence of these measurements indicates that JavaScript is active. More subtly, browsers with a Flash blocking add-on installed show Flash in the `plugins` list, but fail to obtain a list of system fonts via Flash, thereby creating a distinctive fingerprint, even though neither measurement (`plugins`, `fonts`) explicitly detects the Flash blocker. Similarly many browsers with forged User Agent strings are distinguished because the other measurements do not comport with the User Agent.[3]

An example of the fingerprint measurements is shown in Table A. In fact, Table A shows the modal fingerprint among browsers that included Flash or Java plugins; it was observed 16 times from 16 distinct IP addresses.

There are many other measurements which could conceivably have been included in a fingerprint. Generally, these were omitted for one of three reasons:

---

were able to detect installations like this because of the appearance of interleaved cookies (A then B then A) with the same fingerprint and IP. Fingerprints that use hardware measurements such as clock skew [5] (see also note 4) would often be able to distinguish amongst these sorts of "cloned" systems.

[2] AJAX is JavaScript that runs inside the browser and sends information back to the server.

[3] We did not set out to systematically study the prevalence of forged User Agents in our data, but in passing we noticed 378 browsers sending iPhone User Agents but with Flash player plugins installed (the iPhone does not currently support Flash), and 72 browsers that identified themselves as Firefox but supported Internet Explorer userData supercookies.

Variable	Source	Remarks
User Agent	Transmitted by HTTP, logged by server	Contains Browser micro-version, OS version, language, toolbars and sometimes other info.
HTTP ACCEPT headers	Transmitted by HTTP, logged by server	
Cookies enabled?	Inferred in HTTP, logged by server	
Screen resolution	JavaScript AJAX post	
Timezone	JavaScript AJAX post	
Browser plugins, plugin versions and MIME types	JavaScript AJAX post	Sorted before collection. Microsoft Internet Explorer offers no way to enumerate plugins; we used the PluginDetect JavaScript library to check for 8 common plugins on that platform, plus extra code to estimate the Adobe Acrobat Reader version.
System fonts	Flash applet or Java applet, collected by JavaScript/AJAX	Not sorted; see Section 6.4.
Partial supercookie test	JavaScript AJAX post	We did not implement tests for Flash LSO cookies, Silverlight cookies, HTML 5 databases, or DOM globalStorage.

Table 1. Browser measurements included in Panopticlick Fingerprints

1. We were unaware of the measurement, or lacked the time to implement it correctly — including the full use of Microsoft's ActiveX and Silverlight APIs to collect fingerprintable measures (which include CPU type and many other details); detection of more plugins in Internet Explorer; tests for other kinds of supercookies; detection of system fonts by CSS introspection, even when Flash and Java are absent [13]; the order in which browsers send HTTP headers; variation in HTTP Accept headers across requests for different content types; clock skew measurements; TCP stack fingerprinting [14]; and a wide range of subtle JavaScript behavioural tests that may indicate both browser add-ons and true browser versions [15].
2. We did not believe that the measurement would be sufficiently stable within a given browser — including geolocation, IP addresses (either yours or your gateway's) as detected using Flash or Java, and the CSS history detection hack [16].
3. The measurement requires consent from the user before being collectable — for instance, Google Gears supercookie support or the wireless router–based geolocation features included in recent browsers [17] (which are also non-constant).

In general, it should be assumed that commercial browser fingerprinting services would not have omitted measurements for reason 1 above, and that as a result, commercial fingerprinting methods would be more powerful than the one studied here.[4]

3.2 Mathematical Treatment

Suppose that we have a browser fingerprinting algorithm $F(\cdot)$, such that when new browser installations x come into being, the outputs of $F(x)$ upon them follow a discrete probability density function $P(f_n)$, $n \in [0, 1, .., N]$.[5] Recall that the "self-information" or " surprisal" of a particular output from the algorithm is given by:

$$I\big(F(x) = f_n\big) = -\log_2\big(P(f_n)\big), \tag{1}$$

The surprisal I is measured here in units of bits, as a result of the choice of 2 as the logarithm base. The *entropy* of the distribution $P(f_n)$ is the expected value of the surprisal over all browsers, given by:

$$H(F) = -\sum_{n=0}^{N} P(f_n)\log_2\big(P(f_n)\big) \tag{2}$$

Surprisal can be thought of as an amount of information about the identity of the object that is being fingerprinted, where each bit of information cuts the number of possibilities in half. If a website is regularly visited with equal probability by a set of X different browsers, we would intuitively estimate that a particular browser $x \in X$ would be uniquely recognisable if $I\big(F(x)\big) \gtrapprox log_2|X|$. The binomial distribution could be applied to replace this intuition with proper confidence intervals, but it turns out that with real fingerprints, much bigger uncertainties arise with our estimates of $P(f_n)$, at least when trying to answer

[4] While this paper was under review, we were sent a quote from a Gartner report on fingerprinting services that stated,

> Arcot... claims it is able to ascertain PC clock processor speed, along with more-common browser factors to help identify a device. 41st Parameter looks at more than 100 parameters, and at the core of its algorithm is a time differential parameter that measures the time difference between a user's PC (down to the millisecond) and a server's PC. ThreatMetrix claims that it can detect irregularities in the TCP/IP stack and can pierce through proxy servers... Iovation provides device tagging (through LSOs) and clientless [fingerprinting], and is best distinguished by its reputation database, which has data on millions of PCs.

[5] Real browser fingerprints are the result of decentralised decisions by software developers, software users, and occasionally, technical accident. It is not obvious what the set of possible values is, or even how large that set is. Although it is finite, the set is large and sparse, with all of the attendant problems for privacy that that poses [18].

questions about which browsers are uniquely recognisable. This topic will be reprised in Section 4.1, after more details on our methodology and results.

In the case of a fingerprint formed by combining several different measurements $F_s(\cdot), s \in S$, it is meaningful to talk about the surprisal of any particular measurement, and to define entropy for that component of the fingerprint accordingly:

$$I_s(f_{n,s}) = -\log_2\left(P(f_{n,s})\right) \tag{3}$$

$$H_s(F_s) = -\sum_{n=0}^{N} P(f_{s,n}) \log_2\left(P(f_{s,n})\right) \tag{4}$$

Note that the surprisal of two fingerprint components F_s and F_t can only be added linearly if the two variables are statistically independent, which tends not to be the case. Instead, conditional self-information must be used:

$$I_{s+t}(f_{n,s}, f_{n,t}) = -\log_2\left(P(f_{n,s} \mid f_{n,t})\right) \tag{5}$$

Cases like the identification of a Flash blocker by combination of separate plugin and font measurements (see Section 3.1) are predicted accordingly, because $P($`fonts` = "not detected" | "Flash" $\in$ `plugins`$)$ is very small.

3.3 Data Collection and Preprocessing

We deployed code to collect our fingerprints and report them — along with simple self-information measurements calculated from live fingerprint tallies — at `panopticlick.eff.org`. A large number of people heard about the site through websites like Slashdot, BoingBoing, Lifehacker, Ars Technica, io9, and through social media channels like Twitter, Facebook, Digg and Reddit. The data for this paper was collected between the 27th of January and the 15th of February, 2010.

For each HTTP client that followed the "test me" link at `panopticlick.eff.org`, we recorded the fingerprint, as well as a 3-month persistent HTTP cookie ID (if the browser accepted cookies), an HMAC of the IP address (using a key that we later discarded), and an HMAC of the IP address with the least significant octet erased.

We kept live tallies of each fingerprint, but in order to reduce double-counting, we did not increment the live tally if we had previously seen that precise fingerprint with that precise cookie ID. Before computing the statistics reported throughout this paper, we undertook several further offline preprocessing steps.

Firstly, we excluded a number of our early data points, which had been collected before the diagnosis and correction of some minor bugs in our client side JavaScript and database types. We excluded the records that had been directly affected by these bugs, and (in order to reduce biasing) other records collected while the bugs were present.

Next, we undertook some preprocessing to correct for the fact that some users who blocked, deleted or limited the duration of cookies had been multi-counted

in the live data, while those whose browsers accepted our persistent cookie would not be. We assumed that all browsers with identical fingerprints and identical IP addresses were the same.

There was one exception to the (fingerprint, IP) rule. If a (fingerprint, IP) tuple exhibited "interleaved" cookies, all distinct cookies at that IP were counted as separate instances of that fingerprint. "Interleaved" meant that the same fingerprint was seen from the same IP address first with cookie A, then cookie B, then cookie A again, which would likely indicate that multiple identical systems were operating behind a single firewall. We saw interleaved cookies from 2,585 IP addresses, which was 3.5% of the total number of IP addresses that exhibited either multiple signatures or multiple cookies.

Starting with 1,043,426 hits at the test website, the successive steps described above produced a population of 470,161 fingerprint-instances, with minimal multi-counting, for statistical analysis.

Lastly we considered whether over-counting might occur because of hosts changing IP addresses. We were able to detect such IP changes among cookie-accepting browsers; 14,849 users changed IPs, with their subsequent destinations making up 4.6% of the 321,155 IP addresses from which users accepted cookies. This percentage was small enough to accept it as an error rate; had it been large, we could have reduced the weight of every non-cookie fingerprint by this percentage, in order to counteract the over-counting of non-cookie users who were visiting the site from multiple IPs.

4 Results

The frequency distribution of fingerprints we observed is shown in Figure 1. Were the x axis not logarithmic, it would be a strongly "L"-shaped distribution, with 83.6% in an extremely long tail of unique fingerprints at the bottom right, 8.1% having fingerprints that were fairly "non rare", with anonymity set sizes in our sample of 10, and 8.2% in the joint of the L-curve, with fingerprints that were seen between 2 and 9 times.

Figure 2 shows the distribution of surprisal for different browsers. In general, modern desktop browsers fare very poorly, and around 90% of these are unique. The least unique desktop browsers often have JavaScript disabled (perhaps via NoScript). iPhone and Android browsers are significantly more uniform and harder to fingerprint than desktop browsers; for the time being, these smartphones do not have the variety of plugins seen on desktop systems.[6] Sadly, iPhones and Androids lack good cookie control options like session-cookies-only or blacklists, so their users are eminently trackable by non-fingerprint means.

Figure 3 shows the sizes of the anonymity sets that would be induced if each of our eight measurements were used as a fingerprint on its own. In general, plugins and fonts are the most identifying metrics, followed by User Agent,

[6] Android and iPhone fonts are also hard to detect for the time being, so these are also less fingerprintable

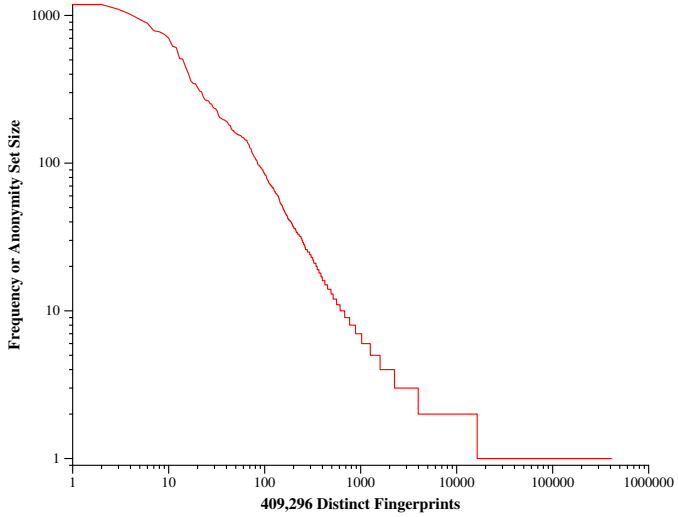

Fig. 1. The observed distribution of fingerprints is extremely skewed, with 83.6% of fingerprints lying in the tail on the right.

HTTP Accept, and screen resolution, though all of the metrics are uniquely identifying in some cases.

4.1 Global Uniqueness

We know that in the particular sample of browsers observed by Panopticlick, 83.6% had unique fingerprints. But we might be interested in the question of what percentage of browsers in existence are unique, regardless of whether they visited our test website.

Mayer has argued [8] that it is almost impossible to reach any conclusions about the *global* uniqueness of a browser fingerprint, because the multinomial theorem indicates that the maximum likelihood for the probability of any fingerprint that was unique in a sample of size N is:

$$P(f_i) = \frac{1}{N} \tag{6}$$

A fingerprint with this probability would be *far* from unique in the global set of browsers G, because $G \gg N$. This may indeed be the maximum subjective likelihood for any single fingerprint that we observe, but in fact, this conclusion is wildly over-optimistic for privacy. If the probability of each unique fingerprint in the sample N had been $\frac{1}{N}$, the applying the multinomial expansion for those 392,938 events of probabilty $\frac{1}{N}$, it would have been inordinately unlikely that we would have seen each of these events precisely once. Essentially, the maximum likelihood approach has assigned a probability of zero for all fingerprints that

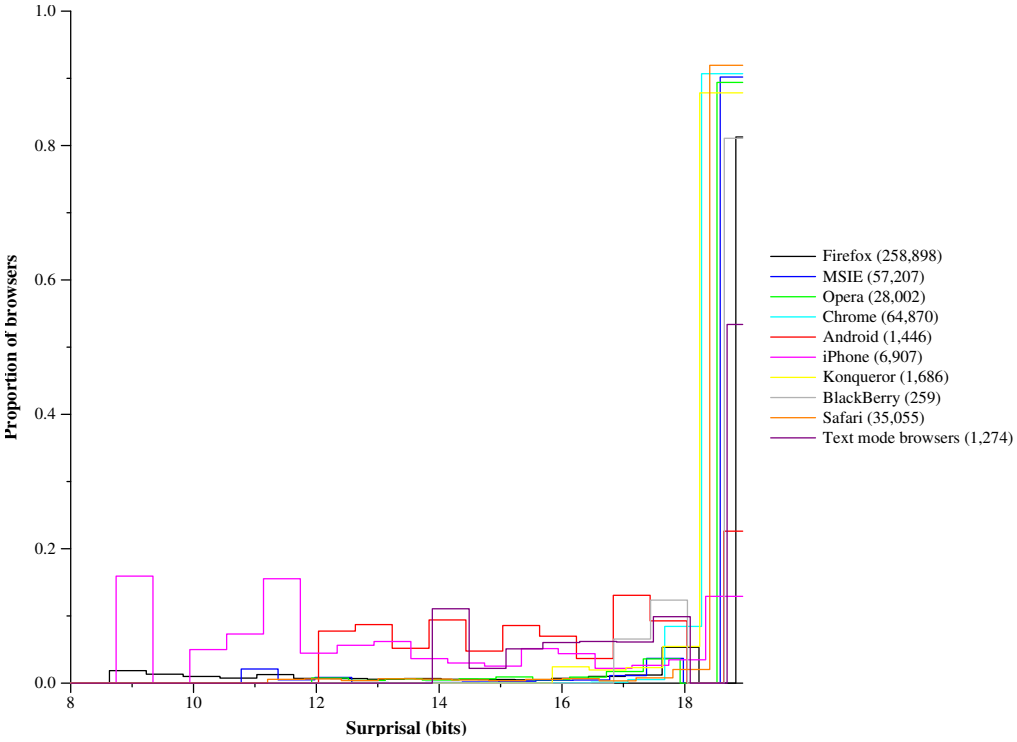

Fig. 2. Surprisal distributions for different categories of browser (believing the User Agent naively; see note 3).

were not seen in the sample N, when in fact many new fingerprints would appear in a larger sample G.

What we could attempt to meaningfully infer is the global *proportion* of uniqueness. The best way to do that would be to fit a very-long-tailed probability density function so that it reasonably predicts Figure 1. Then, we could employ Monte Carlo simulations to estimate levels of uniqueness and fingerprint entropy in a global population of any given size G. Furthermore, this method could offer confidence intervals for the proposition that a fingerprint unique in N would remain unique in G.

We did not prioritise conducting that analysis for a fairly prosaic reason: the dataset collected at `panopticlick.eff.org` is so biased towards technically educated and privacy-conscious users that it is somewhat meaningless to extrapolate it out to a global population size. If other fingerprint datasets are collected that do not suffer from this level of bias, it may be interesting to extrapolate from those.

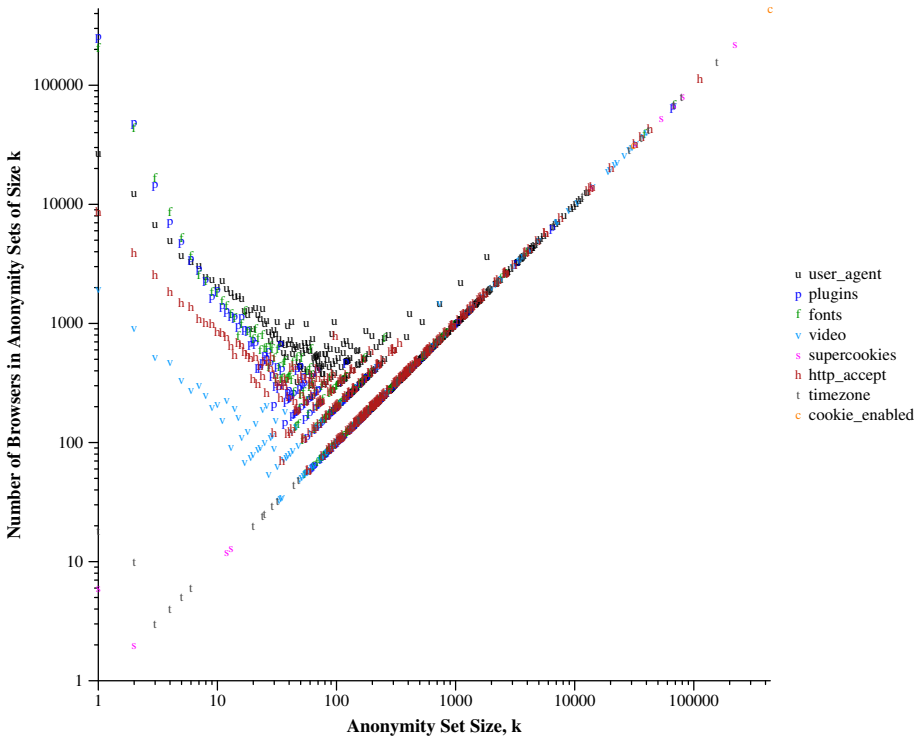

Fig. 3. Number of users in anonymity sets of different sizes, considering each variable separately.

5 How Stable are Browser Fingerprints?

Many events can cause a browser fingerprint to change. In the case of the algorithm we deployed, those events include upgrades to the browser, upgrading a plugin, disabling cookies, installing a new font or an external application which includes fonts, or connecting an external monitor which alters the screen resolution.

By collecting other tracking information alongside fingerprints, we were able to observe how constant or changeable fingerprints were among Panopticlick users. In particular, we used cookies to recognise browsers that were returning visitors, and checked to see whether their fingerprints had changed.

Our observations probably overstate the rate at which fingerprints change in the real world, because the interactive nature of the Panopticlick website encourages to experiment with alterations to their browser configuration.

5.1 Changing Fingerprints as a Function of Time

Among our userbase, rates of fingerprint change for returning cookie-accepting users were very high, with 37.4% of users who visited the site more than once[7] exhibiting more than one fingerprint over time.

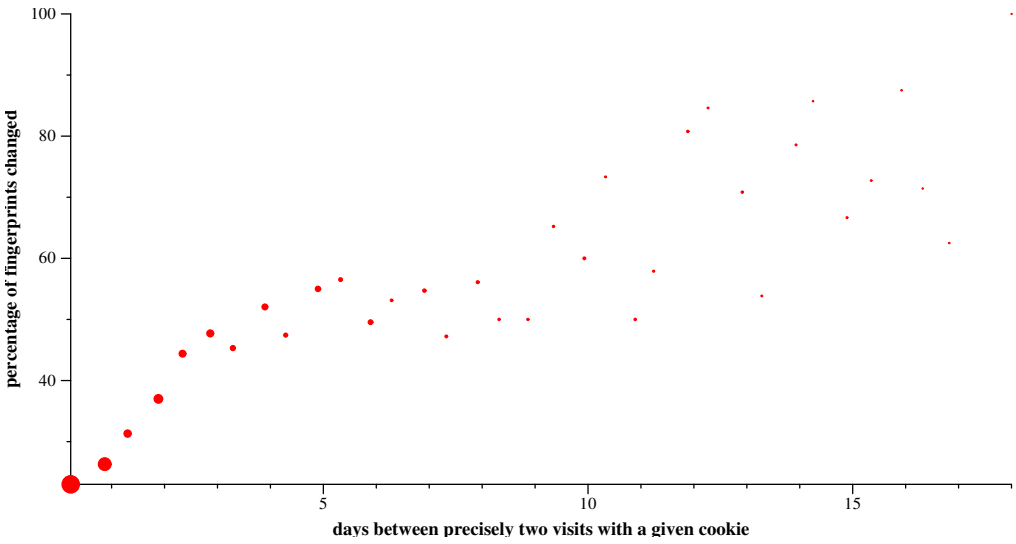

Fig. 4. Proportion of fingerprints that change over given intervals
(area of datapoints indicates number of observations encompassed, $N = 4{,}638$)

The time-dependence of fingerprint changes is illustrated in Figure 4, which plots the proportion of fingerprints that was constant among cookies that were seen by Panopticlick exactly twice, with a substantial time interval in between. The population with precisely two time-separated hits was selected because this group is significantly less likely to be actively *trying* to alter their browser fingerprints (we assume that most people experimenting in order to make their browsers unique will reload the page promptly at some point).

Upon first examination, the high rate of change for fingerprints — even if it overstates the rate of change in the wider Internet population — appears to constitute a powerful protection against fingerprinting attacks.

5.2 Following changing fingerprints

We performed a simple test to see whether a connection can be inferred between the old and new values of fingerprints that change over time.

[7] Our measure of returning visitors was based on cookies, and did not count reloads within 1–2 hours of the first visit.

We implemented a very simple algorithm to heuristically estimate whether a given fingerprint might be an evolved version of a fingerprint seen previously.

The algorithm (set out below) operated on an input fingerprint q, where $F_i(g), i \in \{1..8\}$ are the 8 fingerprint components illustrated in Table 3.1, and G is the set of all browsers observed in our dataset. The algorithm did not attempt to guess a preceding fingerprint if q indicated that the browser did not have Flash or Java installed.

Algorithm 1 guesses which other fingerprint might have changed into q

```
candidates ← [ ]
for all g ∈ G do
  for i ∈ {1..8} do
    if for all j ∈ {1..8}, j ≠ i : F_j(g) = F_j(q) then
      candidates ← candidates + (g, j)
    end if
  end for
end for
if length(candidates) = 1 then
  g, j ← candidates[0]
  if j ∈ {cookies?, video, timezone, supercookies} then
    return g
  else
    # j ∈ {user_agent, http_accept, plugins, fonts}
    if SequenceMatcher(F_j(g), F_j(q)).ratio() < 0.85 then
      return g
    end if
  end if
end if
return NULL
```

`difflib.SequenceMatcher().ratio()` is a Python standard library function for estimating the similarity of strings. We used Python 2.5.4.

We ran our algorithm over the set of users whose cookies indicated that they were returning to the site 1–2 hours or more after their first visit, and who now had a different fingerprint. Excluding users whose fingerprints changed because they disabled javascript (a common case in response to visiting `panopticlick.eff.org`, but perhaps not so common in the real world), our heuristic made a correct guess in 65% of cases, an incorrect guess in 0.56% of cases, and no guess in 35% of cases. 99.1% of guesses were correct, while the false positive rate was 0.86%. Our algorithm was clearly very crude, and no doubt could be significantly improved with effort.

6 Defending Against Fingerprinting

6.1 The Paradox of Fingerprintable Privacy Enhancing Technologies

Sometimes, technologies intended to enhance user privacy turn out to make fingerprinting easier. Extreme examples include many forms of User Agent spoofing (see note 3) and Flash blocking browser extensions, as discussed in Section 3.1. The paradox, essentially, is that many kinds of measures to make a device harder to fingerprint are themselves distinctive unless a lot of other people also take them.

Examples of measures that might be intended to improve privacy but which appear to be ineffective or even potentially counterproductive in the face of fingerprinting include Flash blocking (the mean surprisal of browsers with Flash blockers is 18.7), and User Agent alteration (see note 3). A small group of users had "Privoxy" in their User Agent strings; those User Agents alone averaged 15.5 bits of surprisal. All 7 users of the purportedly privacy-enhancing "Browzar" browser were unique in our dataset.

There are some commendable exceptions to this paradox. TorButton has evolved to give considerable thought to fingerprint resistance [19] and may be receiving the levels of scrutiny necessary to succeed in that project [15]. NoScript is a useful privacy enhancing technology that seems to reduce fingerprintability.[8]

6.2 Enumeratable Characteristics vs Testable Characteristics

One significant API choice that several plugin and browser vendors made, which strengthens fingerprints tremendously, is offering function calls that enumerate large amounts of system information. The `navigator.plugins` object is one example, as are the font lists returned by Flash and Java. Microsoft Internet Explorer deserves an honourable mention for not allowing plugin enumeration, and even though we collected version numbers for $8\frac{1}{2}$ plugins,[9] the plugin entropy on IE was 16.5 bits, somewhat lower than the 17.7 seen in non-IE browsers.

The benefits of allowing Java and Flash to read exhaustive system font lists is questionable. Any website that cares whether someone has the "False Positive BRK" font installed[10] could surely test for it explicitly.

There are probably stronger ease-of-development arguments for making plugins enumeratable, but the example of IE shows that it is not strictly necessary. We recommend that browsers switch to confirm-only testing for fonts and plugins, with an exponential backoff to prevent exhaustive searches by malicious javascript.

[8] We did not try to devise a detection method for NoScript, though they probably exist if users allow scripts from certain important domains.

[9] Our version numbers for Acrobat were approximate and limited to the major version number.

[10] We noticed that font while grepping through the output of one of our analysis scripts.

6.3 Fingerprintability ∝ Debuggability

Much of the entropy we observe in browsers comes from the precise micro-version numbers of all of their plugins. This is somewhat true even in IE, where we were limited to testing the version numbers of $8\frac{1}{2}$ common plugins using PluginDetect and custom JavaScript. A similar, though less severe, problem comes from precise micro-version numbers in User Agent strings.

The obvious solution to this problem would be to make the version numbers less precise. Why report `Java 1.6.0_17` rather than just `Java 1.6`, or `DivX Web Player 1.4.0.233` rather than just `DivX Web Player 1.4`? The motivation for these precise version numbers appears to be debuggability. Plugin and browser developers want the *option* of occasionally excavating the micro-version numbers of clients when trying to retrospectively diagnose some error that may be present in a particular micro-version of their code. This is an understandable desire, but it should now be clear that this decision trades off the user's privacy against the developer's convenience.

There is a spectrum between extreme debuggability and extreme defense against fingerprinting, and current browsers choose a point in that spectrum close to the debuggability extreme. Perhaps this should change, especially when users enter "private browsing" modes.

6.4 Font Orders As An Unnecessary Source of Entropy

When implementing our fingerprinting code, we observed that Adobe Flash and Sun's Java VM not only report complete lists of fonts installed on a system, but return them in non-sorted order, perhaps due to a filesystem inode walk.

We tested the hypothesis that font orders are informative, by checking to see if any returning, cookie-accepting users had font lists whose order had changed. We found that only 30 returning browsers had font lists that were different solely with respect to order. Interestingly, these font lists only varied in the ordering of certain fonts from the "Lucida" family, and there was a related population of about 200 browsers where the same fonts varied in ordering and surrounding whitespace. All of these browsers had Mac OS X User Agent strings, so we concluded that some application on OS X overwrites these font files, either during upgrades or at other times. Aside from this group, our hypothesis that font list orderings were stable turned out to be correct.

Next, we investigated whether a substantial reduction in font list entropy could be achieved if plugins like Flash and Java began sorting these lists before returning them via their APIs. Among browsers where the fonts were detectable, the entropy of the `fonts` variable was 17.1 bits. We recalculated this quantity after sorting to be 16.0, a decrease of only 1.1 bits. Confounding this calculation slightly is the fact that the maximum possible entropy we could detect for either of these numbers, given our dataset, was only 18.4. It is possible that sorting the font lists would have made a much larger difference if the sample size had been large enough for the font entropy and its conceivable ceiling to diverge further.

In contrast to the font case, our pre-launch testing seemed to indicate that the ordering of `navigator.plugins` was not stable in all browsers, so, as noted in Table 3.1, we sorted the plugin list before recording it. We subsequently read Jonathan Mayer's claims that Mozilla actually exposes two different plugin orderings based on *different* inode timestamps [8]. Unfortunately, having sorted our plugin dataset, we cannot test his claims.

7 Conclusions

We implemented and tested one particular browser fingerprinting method. It appeared, in general, to be very effective, though as noted in Section 3.1 there are many measurements that could be added to strengthn it.

Browser fingerprinting is a powerful technique, and fingerprints must be considered alongside cookies, IP addresses and supercookies when we discuss web privacy and user trackability. Although fingerprints turn out not to be particularly stable, browsers reveal so much version and configuration information that they remain overwhelmingly trackable. There are implications both for privacy policy and technical design.

Policymakers should start treating fingerprintable records as potentially personally identifiable, and set limits on the durations for which they can be associated with identities and sensitive logs like clickstreams and search terms.

The Tor project is noteworthy for already considering and designing against fingerprintability. Other software that purports to protect web surfers' privacy should do likewise, and we hope that the test site at `panopticlick.eff.org` may prove useful for this purpose. Browser developers should also consider what they can do to reduce fingerprintability, particularly at the JavaScript API level.

We identified only three groups of browser with comparatively good resistance to fingerprinting: those that block JavaScript, those that use TorButton, and certain types of smartphone. It is possible that other such categories exist in our data. Cloned machines behind firewalls are fairly resistant to our algorithm, but would not be resistant to fingerprints that measure clock skew or other hardware characteristics.

References

1. Lukáš, J., Fridrich, J., Goljan, M.: Digital camera identification from sensor pattern noise. IEEE Transactions on Information Forensics and Security **1**(2) (2006) 205–214
2. Kai San Choi, E.Y.L., Wong, K.K.: Source Camera Identification Using Footprints from Lens Aberration. In: Proc. of SPIE-IS&T Electronic Imaging. Volume 6069., SPIE (2006)
3. Hilton, O.: The Complexities of Identifying the Modern Typewriter. Journal of Forensic Sciences **17**(2) (1972)
4. Kohno, T., Broido, A., Claffy, K.: Remote Physical Device Fingerprinting. IEEE Transactions on Dependable and Secure Computing **2**(2) (2005) 108

5. Murdoch, S.: Hot or not: Revealing hidden services by their clock skew. In: Proc. 13th ACM conference on Computer and Communications Security, ACM (2006) 36
6. The 41st Parameter: PCPrint™ (2008) http://www.the41st.com/land/DeviceID.asp.
7. Mills, E.: Device identification in online banking is privacy threat, expert says. CNET News (April 2009)
8. Mayer, J.: "Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0. Undergraduate Senior Thesis, Princeton University (2009)
9. Krishnamurthy, B., Wills, C.: Generating a privacy footprint on the Internet. In: Proc. ACM SIGCOMM Internet Measurement Conference, ACM (2006)
10. McKinkley, K.: Cleaning Up After Cookies. iSec Partners White Paper (2008)
11. Pool, M.B.: Meantime: non-consensual HTTP user tracking using caches. (2000) http://sourcefroge.net/projects/meantime/.
12. Soltani, A., Canty, S., Mayo, Q., Thomas, L., Hoofnagle, C.: Flash Cookies and Privacy. SSRN preprint (August 2009) http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1446862.
13. Robinson, S.: Flipping Typical (demonstration of CSS font detection). (2009) http://flippingtypical.com/.
14. : TCP/IP stack fingerprinting http://en.wikipedia.org/wiki/TCP/IP_stack_fingerprinting.
15. Fleischer, G.: Attacking Tor at the Application Layer. Presentation at DEFCON 17 (2009) http://pseudo-flaw.net/content/defcon/.
16. : CSS history hack demonstration http://www.whattheinternetknowsaboutyou.com/.
17. W3C: Geolocation API http://en.wikipedia.org/wiki/W3C_Geolocation_API.
18. Narayanan, A., Shmatikov, V.: Robust De-anonymization of Large Sparse Datasets. **2**(2) (2008) 108
19. Perry, M.: Torbutton Design Doccumentation (2009) https://www.torproject.org/torbutton/design.

A Appendix : Some Dataset Queries of Interest

Variable	**Entropy (bits)**
user_agent	10.0
plugins	15.4
fonts	13.9
video	4.83
supercookies	2.12
http_accept	6.09
timezone	3.04
cookies_enabled	0.353

Table 2. Mean surprisal for each variable in isolation

Variable	Value
User Agent	Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.1.7) Gecko/20100106 Ubuntu/9.10 (karmic) Firefox/3.5.7
HTTP ACCEPT headers	text/html, */* ISO-8859-1,utf-8;q=0.7,*;q=0.7 gzip,deflate en-us,en;q=0.5
Cookies enabled?	Yes
Screen resolution	1280x800x24
Timezone	300
Browser plugins	Plugin 0: DivX Web Player; DivX Web Player version 1.4.0.233; libtotem-mully-plugin.so; (AVI video; video/divx; divx). Plugin 1: QuickTime Plug-in 7.2.0; The Totem 2.28.2 plugin handles video and audio streams.; libtotem-narrowspace-plugin.so; (QuickTime video; video/quicktime; mov) (MPEG-4 video; video/mp4; mp4) (MacPaint Bitmap image; image/x-macpaint; pntg) (Macintosh Quickdraw/PICT drawing; image/x-quicktime; pict, pict1, pict2) (MPEG-4 video; video/x-m4v; m4v). Plugin 2: Shockwave Flash; Shockwave Flash 10.0 r42; libflashplayer.so; (Shockwave Flash; application/x-shockwave-flash; swf) (FutureSplash Player; application/futuresplash; spl). Plugin 3: VLC Multimedia Plugin (compatible Totem 2.28.2); The Totem 2.28.2 plugin handles video and audio streams.; libtotem-cone-plugin.so; (VLC Multimedia Plugin; application/x-vlc-plugin;) (VLC Multimedia Plugin; application/vlc;) (VLC Multimedia Plugin; video/x-google-vlc-plugin;) (Ogg multimedia file; application/x-ogg; ogg) (Ogg multimedia file; application/ogg; ogg) (Ogg Audio; audio/ogg; oga) (Ogg Audio; audio/x-ogg; ogg) (Ogg Video; video/ogg; ogv) (Ogg Video; video/x-ogg; ogg) (Annodex exchange format; application/annodex; anx) (Annodex Audio; audio/annodex; axa) (Annodex Video; video/annodex; axv) (MPEG video; video/mpeg; mpg, mpeg, mpe) (WAV audio; audio/wav; wav) (WAV audio; audio/x-wav; wav) (MP3 audio; audio/mpeg; mp3) (NullSoft video; application/x-nsv-vp3-mp3; nsv) (Flash video; video/flv; flv) (Totem Multimedia plugin; application/x-totem-plugin;). Plugin 4: Windows Media Player Plug-in 10 (compatible; Totem); The Totem 2.28.2 plugin handles video and audio streams.; libtotem-gmp-plugin.so; (AVI video; application/x-mplayer2; avi, wma, wmv) (ASF video; video/x-ms-asf-plugin; asf, wmv) (AVI video; video/x-msvideo; asf, wmv) (ASF video; video/x-ms-asf; asf) (Windows Media video; video/x-ms-wmv; wmv) (Windows Media video; video/x-wmv; wmv) (Windows Media video; video/x-ms-wvx; wmv) (Windows Media video; video/x-ms-wm; wmv) (Windows Media video; video/x-ms-wmp; wmv) (Windows Media video; application/x-ms-wms; wms) (Windows Media video; application/x-ms-wmp; wmp) (Microsoft ASX playlist; application/asx; asx) (Windows Media audio; audio/x-ms-wma; wma).
System fonts	wasy10, UnDotum, Century Schoolbook L, OpenSymbol, msam10, Mukti Narrow, Vemana2000, KacstQurn, Umpush, DejaVu Sans Mono, Purisa, msbm10, KacstBook, KacstLetter, cmr10, Norasi, Loma, KacstDigital, KacstTitleL, mry_KacstQurn, URW Palladio L, Phetsarath OT, Sawasdee, Tlwg Typist, URW Gothic L, Dingbats, URW Chancery L, FreeSerif, ori1Uni, KacstOffice, DejaVu Sans, VL Gothic, Kinnari, KacstArt, TlwgMono, Lohit Punjabi, Symbol, Bitstream Charter, KacstOne, Courier 10 Pitch, cmmi10, WenQuanYi Zen Hei Mono, Nimbus Sans L, TlwgTypewriter, VL PGothic, Rachana, Standard Symbols L, Lohit Gujarati, kacstPen, KacstDecorative, Nimbus Mono L, Mallige, Nimbus Roman No9 L, KacstPoster, Mukti Narrow, WenQuanYi Zen Hei, FreeSans, cmex10, KacstNaskh, Lohit Tamil, Tlwg Typo, UnBatang, KacstFarsi, Waree, KacstTitle, Lohit Hindi, DejaVu Serif, Garuda, KacstScreen, FreeMono, URW Bookman L, cmsy10 (via Flash)
(Partial) supercookie tests	DOM localStorage: Yes, DOM sessionStorage: Yes, IE userData: No

Table 3. A typical Panopticlick fingerprint

User Agent	Cookies?	Video, Timezone, Plug-ins, Fonts, Supercookies	Frequency
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7	Yes	no javascript	1186
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Mobile/7D11	No	no javascript	1100
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2) Gecko/20100115 Firefox/3.6	Yes	no javascript	1017
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2) Gecko/20100115 Firefox/3.6	Yes	no javascript	940
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2) Gecko/20100115 Firefox/3.6 (.NET CLR 3.5.30729)	Yes	no javascript	886
Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.2) Gecko/20100115 Firefox/3.6 (.NET CLR 3.5.30729)	Yes	no javascript	788
Mozilla/5.0 (Windows; U; Windows NT 6.1; de; rv:1.9.2) Gecko/20100115 Firefox/3.6	Yes	no javascript	775
Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.2) Gecko/20100115 Firefox/3.6	Yes	no javascript	746
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7	Yes	no javascript	702
Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9.1.7) Gecko/20091221 Firefox/3.5.7 (.NET CLR 3.5.30729)	Yes	no javascript	618

Table 4. 10 Largest Anonymity Sets

User Agent	Cookies?	Video	Timezone	Frequency
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7D11 Safari/528.16	Yes	320x396x32	480	345
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; de-de) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7D11 Safari/528.16	Yes	320x396x32	-60	280
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7D11 Safari/528.16	Yes	320x396x32	360	225
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7D11 Safari/528.16	Yes	320x396x32	0	150
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; de-de) AppleWebKit/528.18 (KHTML, like Gecko) Mobile/7D11	Yes	320x396x32	-60	149
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Mobile/7D11	Yes	320x396x32	480	149
Mozilla/5.0 (iPod; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7D11 Safari/528.16	Yes	320x396x32	300	145
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_2 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Mobile/7D11	Yes	320x396x32	0	114
Mozilla/5.0 (Linux; U; Android 2.0.1; en-us; Droid Build/ESD56) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17	Yes	480x854x32	300	112
Mozilla/5.0 (iPod; U; CPU iPhone OS 3_1_2 like Mac OS X; de-de) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7D11 Safari/528.16	Yes	320x396x32	-60	97

Table 5. 10 Largest Anonymity Sets with Javascript