PlexiWall: Thwarting Paywall Bypassers Using Browser Fingerprinting

Hong-Long (Kit) Nguyen
Harvard University
Cambridge, MA
honglongnguyen@college.harvard.edu

Kyle Fullen
Harvard University
Cambridge, MA
kfullen@college.harvard.edu

Mark Wilkening
Harvard University
Cambridge, MA
wilkening@g.harvard.edu

ABSTRACT

With the advent of the Internet, increasingly more people prefer to get their news online. Unlike print ads, due to their relatively low price online ads often represent an insubstantial portion of revenue for content providers. To monetize digital content businesses have turned to the digital subscription model. The simplest way to deploy a digital subscription model is via a hard paywall, however this significantly reduces readership, especially for general news outlets without niche content and a devoted community to follow it. The soft metered paywall – where a limited quantity of content is provided for free with full access requiring a subscription – is the most popular paywall implementation today that gates two thirds of all digital news subscriptions. Unfortunately, the vast majority of popular metered paywalls can be easily circumvented. In this paper we present PlexiWall, a soft metered paywall implementation which utilizes browser fingerprinting techniques to prevent all existing circumvention methods, allowing content providers to offer free content without leaving themselves vulnerable to abusive bypassing techniques. We show that PlexiWall offers a practical soft metered paywall implementation, with high usability and deployability, as well as strong security and resistance to current bypassing techniques.

KEYWORDS

paywalls, bypassers, fingerprinting, PlexiWall

ACM Reference Format:
Hong-Long (Kit) Nguyen, Kyle Fullen, and Mark Wilkening. 2018. PlexiWall: Thwarting Paywall Bypassers Using Browser Fingerprinting. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

With the advent of the Internet, increasingly more people prefer to get their news online. Pew surveys suggest that at the turn of the century, 45% of Americans get most of their news in print while only 13% do so online [11]. By 2018, these quantities have flipped with over half of US adults often getting their news online while only 16% do so via traditional newspapers [26]. This drastic shift in landscape resulted in thousands of smaller publications going under and major newspapers having to reinvent their business models in order to survive. From 2003 to 2014, US publishers saw the annual spending on their main revenue source of print advertisements free falling from $44.9 billion down to $16.4 billion [10] (see Figure 1). Though the digital news advertisement space did see an uptick from $1.2 billion to $3.5 billion [10] (see Figure 1), due to their relatively low price, online ads often represent an insubstantial portion of revenue, especially for smaller newspapers. Publishers big and small then had to find new sources of revenue, and along came the digital news subscription.

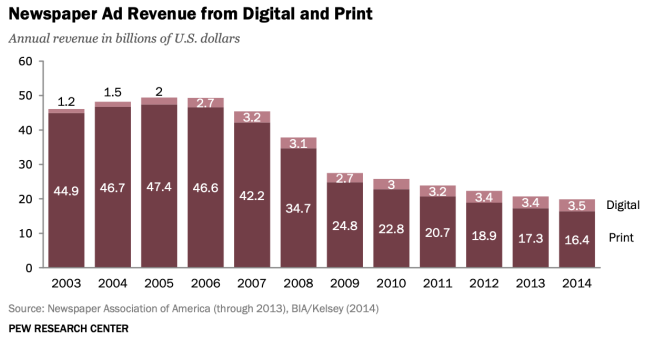

Figure 1: Newspaper Advertising Revenue. Visualization by Pew Research Center [10].

Over two thirds of all major publishers in the US and Europe now charge for some or all of their digital content [27], with smaller newspapers also quickly following suit (see Figure 2). The simplest way to deploy a digital subscription model is via a hard paywall – a setup where all of the publication's content is kept behind a standard username–password authentication barrier. In this scheme, paid subscribers with login credentials get full access, while non-paying visitors are completely barred out. Implementing a hard paywall, however, significantly reduces the readership of the paper because an overwhelming majority of people will rather consume their news elsewhere for free than pay for one specific publication [13]. Primarily due to this reason, only about one sixth of all digital subscriptions are gated by a hard paywall [22]. Publishers who opt for this aggressive gating can usually justify so for they offer in-depth reporting in a niche field like finance (The Wall Street Journal – the hard paywall spearhead since 1997) or sports (The Athletic).

The second one sixth of all digital subscription paywalls out there is made up of hard freemium paywalls [22]. A freemium

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

paywall is technologically almost equivalent to a hard paywall, with the only difference being that it is only applied to a small catalog of premium content. The rest, and the majority, of the articles, are freely available to non-subscribers under this paywall implementation. A common scheme of having free news reporting and premium editorials is appealing among both publishers and readers for the casual reader gets to access their digital news for free from their favorite paper, while the more serious patrons can choose to pay for their favorite op-eds. While attractive in principle, freemium paywalls are not the most popular in practice for they require editorial overhead and yet they do not extract the maximal revenue possible for most readers only care about breaking news, which is usually free. Nevertheless, the freemium model still has significant traction among major, nationally circulated papers in Europe (e.g. Daily Telegraph) [27].

Last, but certainly not least, the soft metered paywall is the most popular paywall implementation that gates two thirds of all digital news subscriptions. Pioneered by The New York Times back in 2011, the metered paywall has now become the go-to monetization implementation for major publications in the US [27]. This type of paywall allows a non-subscribed reader to access up to a limited number (e.g. 10) of articles per month for free. After exhausting the monthly limit, the visitor will be barred from reading any more articles until they either pay for a subscription, or a new month comes around. This design is generally considered most attractive because it allows casual readers, i.e. potential subscribers, to freely access articles they care most about and explore what the publication has to offer. The publisher, on the other hand, gets to charge all regular readers who are willing to pay, regardless of whether they enjoy the weekday briefs or the weekend op-eds.

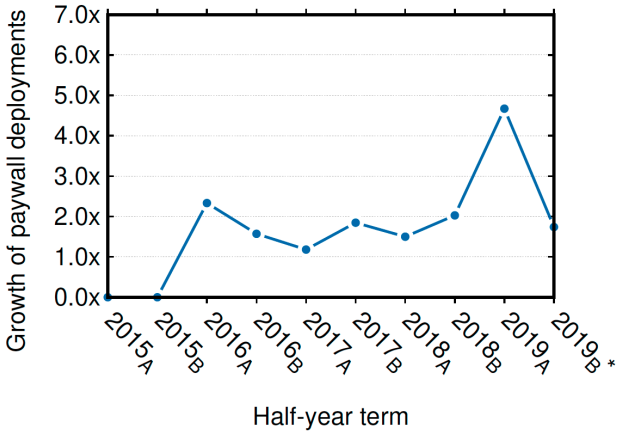

Figure 2: Growth of paywall deployments each six months. Visualization by Papadopoulos et al [22].

In unfortunate news to publishers, however, the vast majority of these popular metered paywalls can be easily circumvented, allowing non-paying readers to freely access all the paywalled content. While later in the paper we will outline all the technical details that allows for a myriad of circumvention techniques, all that a lay Internet user needs to do is to install one of the many freely available paywall circumvention browser extensions. Once they do so, they will get unlimited access to metered paywalled articles from The New York Times, The Washington Post, and The Boston Globe, among others. These "paywall blockers" are steadily gaining in popularity. If/when they gain enough traction like their ad blocking cousins, from the news business' point of view, having such a vulnerable metered paywall would completely undermine the idea of having one at all.

In this paper, we introduce PlexiWall, an impenetrable metered paywall utilizing integrated, content server–managed browser fingerprinting. In sections 2.1 and 2.2, we go over the underlying design of commercial metered paywalls, as well as how vulnerabilities in both their design and active implementations make them easily bypassable. While section 2.3 introduces the technique of browser fingerprinting and previous works regarding it, section 3 outlines our use of browser fingerprinting in the metered PlexiWall. Section 4: evaluation. Section 5: conclusion.

2 RELATED WORK

2.1 Metered Paywall Design

The goal of a metered paywall is to bar readers from accessing more than a prescribed amount of content (e.g. 10 articles/month), unless they subscribe. The paywall then creates an incentive for frequent readers to pay for a subscription, thus generating a precious source of steady revenue for newspapers.

From keeping track of your digital shopping cart to authenticated banking sessions to implicit browsing habits, browser cookies have become pervasive in storing our personal information. Cookies, too, are used by many news websites as the primary metered paywall mechanism that keep tracks of the number of articles each browser has visited in a month thus far. Roughly anywhere between 39% to 75% of all metered paywalls rely on cookies alone as their enforcement mechanism. The first 39% figure is from our analysis of the most popular Bypass Paywalls browser extension covering 162 paywalled sites, while the second 75% figure is from a random sample of 28 soft paywalls analyzed by Papadopoulos et al. [22]. This is immediately concerning for cookies are browser-stored and user-controlled, which means that a shifty user should be able to spoof the paywall by appropriately manipulating the cookie. This is indeed the major flaw in cookie-based metered paywalls and we will explore and exploit this vulnerability more in depth in section 2.2.

More sophisticated soft paywalls do exist and are deployed relatively widely, though their sophistication, unfortunately, does not translate to their security. Here, we will step through high-level mechanisms of the most popular third-party paywall at the moment [22], Piano's Tinypass, whose code is curiously available at http://code.tinypass.com/tinypass.js and https://cdn.tinypass.com/api/libs/fingerprint.js. Note that the discovery of this source code and its first analysis was done by Papadopoulos et al. [22]. Tinypass offers both hard and soft metered paywalls as a service. We will only focus on the soft paywall implementation below:

Step 1: User's browser makes request for content of a website where Tinypass paywall solution is installed.

Step 2: The website sends back JavaScript instructing the browser to request code from Tinypass' servers, as well as the entirety of

the content's page HTML.
Step 3: Per the JavaScript instruction, the browser fetches Tinypass' code libraries;
Step 4: And then executes the Tinypass code, which a) fingerprints the browser, and b) checks for a valid Tinypass cookie. If the cookie exists, Tinypass uses the cookie to track the user's visit counts. Else, Tinypass looks for the browser's fingerprint in its server database and checks whether the fingerprint already exists and has server-stored visit counts associated with it. Note that although Tinypass does use browser fingerprinting, it only utilizes fingerprinting as a secondary tracking mechanism in case the cookie is missing, and its fingerprint look up method is flawed as future steps will reveal.
Step 5: After having collected all the browser associated data, Tinypass code sends the data back to a Tinypass server via a POST request. The server then responds back with a JSON that among others, dictates whether or not the content HTML is displayed to the user.
Step 6: The Tinypass code now finally enforces the paywall policy, potentially preventing the browser and user from accessing the paywalled content.

In spite of their seemingly sophisticated design, commercial metered paywalls relying on known third-party libraries and servers like Piano's Tinypass or the French paywall service Poool can be simply circumvented by disabling JavaScript code from these sites, as Bypass Paywalls effectively does. Critical vulnerabilities like this exist on the vast majority of currently deployed metered paywalls, and how they can be exploited will be the focus of the following subsection.

2.2 Soft Paywall Bypasser

As noted in section 2.1, soft paywalls leverage the use of cookies to keep track of returning users. The reason these paywalls are labeled as soft however is because there are numerous known techniques users can leverage to easily bypass this type of paywall. These techniques are implemented in many open-source bypassers used to bypass these soft paywalls. For example, one open-source bypasser we examined was Bypass Paywalls [8], which essentially clears a user's cookies associated with the site of interest in order to spoof the site into thinking that the returning user is a new user. Besides clearing cookies though, the bypasser utilizes other techniques as well, the usage distribution of which can be seen in Figure 3. These and other techniques for bypassing soft paywalls are outlined and further examined below.

2.2.1 Technique 1: Using Incognito or Private Browsing Mode. One way to bypass some soft paywalls is to simply use incognito or private mode on your browser. Browsing in incognito mode disallows sites to load cookies onto your local machine. Thus some of these soft paywalls, which solely use cookies to track users, are no longer able to track you. All major web browsers come with this capability, and thus, virtually all users are able to bypass some soft paywalls using this technique. According to Papodopoulos et al. [22], in their study this technique could sufficiently bypass most of the soft paywalls tested.

2.2.2 Technique 2: Clearing Cookies. Another way to bypass a soft paywall is to simply clear one's browser cookies. Most browsers

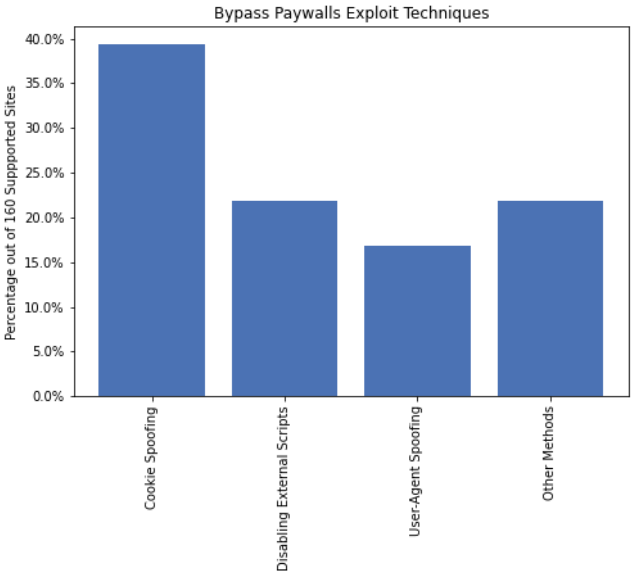

Figure 3: Distribution of paywall exploit techniques used by Bypass Paywalls.

allow for a user to clear their cookies, specifically in their Privacy settings, since cookies are stored on the client. Thus the user can clear their cookies at any time, without notifying the server. After clearing one's cookies, on a client's subsequent return to the site, the server will see that the client has no cookies and will thus perceive the client as a new user. This technique is used in many standard paywall bypassers since many soft paywalls can be bypassed in this manner. In a study conducted by Papadopoulos et al. [22], clearing cookies allowed the bypassing of 75% of soft paywalls tested.

2.2.3 Technique 3: Switching Browsers. Another method is to switch the browser one is using. Each browser stores their own set of cookies. In other words, cookies are not shared across browsers. Thus if a user has cookies for a news site stored for one browser but not for another, the user can simply launch the site from the cookieless browser to have the site perceive them as a new user. This technique is very limited however as there are a limited number of browsers to use and it is inconvenient for the user to have to switch browsers.

2.2.4 Technique 4: Disabling JavaScript. Publishers sometimes use JavaScript to implement their paywalls. A user can disable JavaScript in their browser to prevent the paywall script from being able to run, thus disabling the paywall. Most browsers have the option to disable JavaScript [8], but there are browser extensions that do this as well [7]. This technique can successfully bypass some paywalls but it can also prevent other potentially critical features of the site from working if those features are implemented using JavaScript. There are extensions though such as NoScript [9] that allow the user to choose which JavaScript scripts can run on a webpage, easing the issue of broken site functionality.

2.2.5 Technique 5: Blocking requests for popular paywall libraries and scripts. Many websites rely on third-parties for their implementations. These third-party paywall providers offer "paywall-as-a-service" to publishers, meaning that publishers pay a fee for them to enforce and manage the paywall on these publishers' websites [22]. Additionally, apparently a small number of third-party services account for the vast majority of paywall deployments online [22]. Because of this, large-scale circumvention becomes easier, and thus, one other technique that clients use to bypass paywalls is blocking requests to popular third-party libraries that are used in paywall implementations. In Papodopoulos et al.[22], this technique allowed the bypassing of 48% of soft paywalls without breaking the main functionality of the corresponding sites.

2.2.6 Technique 6: Spoofing the user-agent and referer fields. HTTP request headers contain fields called user-agent and referer. The user-agent field is provided by the browser to help the server identify the user (or user agent), giving information such as which browser is being used, what version, and which operating system. The referer field tells the server the address of the page making the request. Users can change or spoof these fields in their GET requests to the server to prevent sites from obtaining accurate data on them. In fact some publishers allow unrestricted access to content for users coming from large third-party services such as Google Search, Twitter, or Facebook due to pay-for-promotion initiatives [22]. Thus spoofing the referer field to one of the addresses for these services can allow users to bypass paywalls for these sites, including even hard paywalls [22]. In terms of spoofing the user-agent field, Papadopoulos et al. cites that they were able to bypass 12% of soft paywalls using this technique [22].

2.3 Fingerprinting

2.3.1 Overview. In response to the above techniques that users use to avoid tracking, websites have come to use what is called browser fingerprinting in order to identify returning users. Essentially when the user sends a request to access a website, the browser makes available and sends specific data to the receiving server. This includes information about the user's browser, operating system, and device configurations. More specifically, websites are able to have access to information such as a user's screen resolution, system fonts, browser and OS versions, installed plugins or extensions, and timezone just to name a few examples. Websites collect this information and use an algorithm to generate a unique fingerprint in order to identify unique users and track their behavior. In terms of persistence, fingerprints are more ideal than cookies in that they cannot be deleted on the client side and are rendered useless only by a user configuration change large enough such that the algorithm ends up generating a new fingerprint when the user returns to the site. The stability of a fingerprint depends on what information the algorithm uses to generate this fingerprint. For example, a fingerprinting algorithm that heavily relies on a user's browser version to distinguish users can lead to flimsy fingerprints since browser updates can occur relatively often; on the other hand, an algorithm that relies on a user's system fonts, which rarely change, can lead to a more persistent fingerprint. Regardless, there are even crude algorithms that can detect with relatively high probability that a user's fingerprint has changed [15], which helps quell the issue of stability. Fingerprints can even be used to regenerate cookies and can distinguish machines behind a single IP address. Additionally, although manual adjustment of one's device, browser, or OS settings over time can render fingerprints useless, most of these settings are more or less set in stone since users rarely ever change these. Fingerprinting thus proves to be a very useful alternative for identifying unique users.

2.3.2 Related Work. There have been a litany of works dedicated to exploring the effectiveness of fingerprinting and different fingerprinting implementations. Starting in 2009, Mayer first generated the idea of fingerprinting in his thesis paper [18]. Particularly, he investigated whether differences in browsing environments could be exploited by website servers to deanonymize and thus identify users. He realized that browsers could provide information about their own configurations, the operating system, and hardware on which they are running. He conducted an experiment collecting the content of various objects of browsers who connected to the website of his experiment. Specifically, these objects were *navigator*, *screen*, *navigator.plugins*, and *navigator.mimeTypes*. He found that out of the 1328 clients that visited the site, 96% could be uniquely identified. However, the small scale of the experiment prevented him from concluding anything more general or substantial about fingerprinting [17].

In 2010 though Eckersley [15] generated a much larger experiment very similar to that of Mayer through his Panoptoclick website [6]. Through widespread advertisement, he was able to draw in significantly more users to his site than in the Mayer experiment and thus amassed 470,161 fingerprints. To generate these fingerprints he collected measurements from both static HTTP requests and through using AJAX. These measurements were grouped into eight separate strings which were concatenated to form the fingerprints. The sources from which these characteristics or measurements were gathered were the client's user agent string, HTTP headers, enabled cookies setting, screen resolution, time zone, browser plugins, system fonts, and supercookies. The author claims that these measurements and not others (i.e. IP address, geolocation) were primarily chosen due to their being sufficiently stable within a given browser. However, he also claims that he could have included other stable measurements but did not have time to implement the testing suite to gather these measurements correctly. For example, the author claims he could have potentially made use of Microsoft's ActiveX and Siverlight APIs, which collect fingerprintable measurements such as various client CPU and OS information. Regardless he found that with the gathered measurements, he was able to generate a unique fingerprint for 83.6% of users. This paper coined the term "browser fingerprinting" and was first to prove that fingerprinting was viable at a large scale [17].

The Eckersley paper coupled with the development of new browser features motivated future studies to focus on adding new information to use in browser fingerprints and tracking them over time [17]. In 2012 Mowery and Shacham [20] studied the Canvas API, which allows users to draw graphics using the graphical capabilities of their browsers, and its potential for use in fingerprinting. They noticed that font handling stacks can vary between devices. Additionally when a client visits a website, the website can draw an element, using the Canvas API configured with the client's settings,

and generate pixel data from that element. Thus, the hashing of this pixel data they discovered could lead to a unique fingerprint for websites to utilize. Acar et al. [12] generated a larger-scale study of this new method of fingerprinting, which is now known as canvas fingerprinting. They additionally studied the WebGL API, which allows browsers to render and manipulate interactive 3D objects, and its potential for fingerprinting; however, no progress was made on its fingerprinting capabilities until later studies [17].

Another fingerprintable measurement of study includes identifying a user's browser extensions or plugins. Identifying particular plugins is difficult because there is no existing API that can query an exact list of a browser's installed extensions [17]; however, discovering this information can prove very fruitful since plugins are particularly unique to a given user and in fact multiple studies have found alternative ways to detect extensions. Specifically, in an initial study by Sj́osten et. al [28], they were able to use web accessible resources to detect a user's extensions. However they found that they were only able to detect 28% and 7% of all Chrome and Firefox extensions respectively with this technique. A later study [16] utilized Sjösten's technique and concluded that they could generate unique fingerprints for 39% of users, given their gathered extension information. Starov and Nikiforakis [29] exploited side effects produced by browser extensions to detect the use of those respective extensions, finding that they were able to generate unique fingerprints for 14% of users. Lastly Sánchez-Rola et al. [25] used timing side channel attacks to detect extensions, generating unique fingerprints for 57% of users.

One last measurement of note includes gathering CPU and GPU information through benchmarking. Through Javascript, websites can use a script to launch a series of tasks and measure how long it takes to compute them. However, one needs to be wary of how to interpret differences and fluctuations in the measurements as these can be attributed to numerous factors. Mowery et al. [19] used tests to identify the performance characteristics of a browser's Javascript engine. Nakibly et al. [21] benchmarked the GPU and found that this could lead to very noticeable differences between devices. Lastly, Saito et al. in two studies [23] [24] used benchmarking to discover CPU characterisitcs and was able to identify CPU family and number of cores with high accuracy.

Studies have additionally tried to uncover how often fingerprints change over time. Eckersley [15] found that over the course of his study 37.4% of returning users generated more than one fingerprint. Vastel et al. [30] conducted a more extensive study on this issue and found that one change was observed for 45% of collected fingerprints after one day, while other devices' fingerprints took several weeks before seeing a change. However, they were able to track a device for 51.8 days on average. Additionally they were able to track 26% of devices for over 100 days, implying that fingerprinting is still relatively useful for tracking users. Moreover, according to Eckersley [15] there exist algorithms that can detect if a user's fingerprint has changed and that can correctly guess their modified fingerprint. Even with using a notably crude algorithm, Eckersley was able to guess a user's modified fingerprint correctly on 65% of cases.

3 BROWSER FINGERPRINTING PAYWALL IMPLEMENTATION

Our solution for a soft paywall implementation relies on browser fingerprinting to maintain anonymous sessions for free-tier users. We prototype this implementation by modifying an open-source cookie-based paywall implementation [1]. The base implementation is written in Django [2], by extending the existing user authentication libraries [3]. The implementation for authenticated web elements is extended to include content tracking information and record visits to unique content within the Django user session. Django user sessions allow for anonymous users, password authenticated users, and third-party authenticated users. By adding the tracking information free content limits can be imposed when non-authenticated anonymous users attempt to access new content while over the specified content limit.

The problem with the base implementation is straightforward. Django authentication sessions rely on cookies within the client to maintain session identifiers and request sequence numbers. A user can choose to terminate a session at any point by clearing their browser cookies. For anonymous sessions, this resets the paywall tracking information, allowing for a fresh set of free content.

To address this problem, we need to be able to maintain persistent tracking information for users outside of the standard transient sessions, and have some way of uniquely-identifying anonymous users such that all of their sessions can be pinned to the persistent tracking information. We do this by storing tracking information in a key-value store within the webserver, and use a browser fingerprinting implementation to provide unique identifiers for indexing the persistent session information. We use the open-source BFA package [4], a Django wrapper around the open-source fingerprintJS implementation [5]. Because accessing the browser fingerprint requires a JavaScript invocation, users must now explicitly log-in to anonymous sessions (via a simple button click) before accessing free content. The anonymous log-in must be processed using TLS (i.e. requiring HTTPS), such that new sessions cannot be made using spoofed fingerprints.

3.1 Overview of Fingerprinting Implementation

The fingerprintJS browser fingerprinting implementation sources a wide variety of browser specific information to ultimately generate a unique SHA256 hash value representing an individual browser instance. At a high level there are two basic methods for sourcing browser identifying information: (1) environment and configuration information and (2) variation in complex processing tasks. Table 1 presents a complete list of the information used within the fingerprintJS implementation.

The first is the most obvious, simply query the browser for self reported environment and configuration information. Modern browsers contain a vast quantity of configurable state, and even this directly accessible information can be a strongly differentiating signal. Information such as software versions of libraries, hardware identifiers and parameters, installed content such as plug-in information and fonts, and user settings such as time-zone or language are all directly accessible through Javascript in the browser.

Table 1: FingerprintJS Browser Information Sources

Environment/Version	Computational Tasks
Adblock / Plugins	Audio Test
Screen Resolution	Canvas Rendering
Browser Version	Device Memory/ Local Storage
Color Depth	Error Handling Implementation
Cookie Blocking	
CPU Class	
Hardware Concurrency	
Touch Support	
Platform String	
Fonts	
Languages	
Timezone	

Table 2: PlexiWall Usability and Deployability

Usability		Deployability	
Memorywise-Effortless	✓	Accessible	✓
Scalable-for-Users	✓	Negligible-Cost-per-User	✓
Nothing-to-Carry	✓	Server-Compatible	✓
Physically-Effortless	✓	Browser-Compatible	✓
Easy-to-Learn	✓	Mature	✗
Efficient-to-Use	✓	Non-Proprietary	✗
Infrequent-Errors	~		
Easy-Recovery-from-Loss	✓		

The second category of fingerprinting information leverages the minor differences in lower level software and hardware for unique browser instances which influence complex processing tasks. Prime examples in this category include graphics processing tasks such as rendering a canvas containing complex overlapping objects, or audio processing tasks such as rendering a compressed audio signal. These tasks operate on relatively large amounts of data and can be influenced by minor differences in supporting libraries, compilers and optimizations, and hardware acceleration features.

4 EVALUATION

For content providers, PlexiWall provides a secure metered paywall. This is accomplished by introducing in essence, an anonymous client authentication solution. For our evaluation, we consider both the usability and deployability of this authentication method, as well as the overall security PlexiWall provides to protect content providers.

4.1 Usability and Deployability

We evaluate PlexiWall using the usability and deployability criteria from Bonneau et al.[14], reproduced in Table 2. PlexiWall fully meets 7 out of the 8 usability criteria. We classify PlexiWall with *Quasi-Infrequent-Errors*, because although fingerpinting hash collisions can prevent anonymous users from accessing their full allocated free content, these collisions are rare with high quality fingerprinting and never allow improper access to paid content.

Table 3: Susceptibility to Bypassing Techniques

Incognito or Private Browsing	✗
Clearing Cookies	✗
Switching Browsers	✓
Disabling JavaScript	✗
Blocking Third-Party Requests	✗
Spoofing User-agent / Referer	✗

Although this error may result in user frustration, the technique enables the content provider to provide the free content in the first place, and for the anonymous user, receiving fewer free articles in error is still better than a hard paywall with none at all.

In the deployability department, PlexiWall fully meets 4 out of the 6 criteria. At the moment, PlexiWall is not *Non-Proprietary* since for commercial deployment, it will likely require a more complex browser fingerprinting scheme like the one offered by the commercial version of fingerprintJS. Lastly, PlexiWall is not *Mature*. Though every individual component of the PlexiWall system has been relatively well researched, our implementation, to the best of our knowledge, is the first browser fingerprinting metered paywall of its kind that claims the security guarantees below.

Though it has very impressive usability and deployability benefits, many of them derive directly from the virtue of being a metered paywall. A traditional metered paywall also fully meets all of the usability criteria, and out of the deployability ones, it only misses out on *Non-Proprietary*. Contemporary paywalls are also only *Quasi-Non-Proprietary* because although a thorough internet crawl can find you a few open-source metered paywall implementations, they are all scrappy toy examples at best, and the money to be made from the now-established "paywall-as-a-service" space will likely discourage future open-source work in this sector. The inevitable drop in the *Mature* category aside, PlexiWall is then very user-friendly and practical to deploy just like a traditional metered paywall.

4.2 Security

Where PlexiWall distinguishes itself is in its superior security assurances when compared to metered paywalls deployed today. Table 3 presents an overview of the susceptibility of PlexiWall to conventional paywall bypassing techniques, described previously in Section 2.2. Cookie-blocking techniques will be detected and result in blocked access, while cookie-clearing techniques will not effect the persistent session information stored on the web-server. After clearing cookies the user will need to restart an anonymous session to access content. Like blocking cookies, blocking JavaScript will be detected and result in blocked access. PlexiWall does not rely on third party providers to enforce the paywall. Although most of the site can be served over HTTP, starting an anonymous session is forcibly required to use HTTPS, preventing the spoofing of fingerprinting information, or any fields used when starting an anonymous session. Out of the ones that work can circumvent soft paywalls, the only bypassing technique which can work for PlexiWall is to switch browsers, because our solution fundamentally associates one user with one browser. This is inconvenient for

users and results in a fairly limited number of additional anonymous sessions. Circumvention techniques targeted specifically at randomizing the browser's fingerprint exist, like utilizing the Tor or Brave browsers, but these methods also could be ultimately detected by a better fingerprinting solution [17], and so we will leave these techniques outside the scope of our discussion.

5 CONCLUSION

The soft metered paywall is the most popular paywall implementation today that gates two thirds of all digital news subscriptions. Current implementations however are easily bypassed with simple techniques allowing users to abusively ignore meter restrictions. In this paper we have presented PlexiWall, a soft metered paywall implementation which utilizes browser fingerprinting techniques to prevent meter circumvention. PlexiWall allows content providers to offer free content without leaving themselves vulnerable to abusive bypassing techniques. We show that PlexiWall offers a practical soft metered paywall implementation, with high usability and deployability, as well as strong security and resistance to current bypassing techniques. This benefits both content providers and users, as content providers can have assurances about rightful access to their content, while users can enjoy the free content available from the soft paywall model and not be forced back behind hard paywall subscriptions.

REFERENCES

[1] [n.d.]. https://github.com/richardcornish/django-registrationwall
[2] [n.d.]. https://www.djangoproject.com/
[3] [n.d.]. https://docs.djangoproject.com/en/3.1/topics/auth/
[4] [n.d.]. https://pypi.org/project/bfa/
[5] [n.d.]. https://github.com/fingerprintjs/fingerprintjs
[6] [n.d.]. Covering Your Tracks website. https://panopticlick.eff.org/
[7] [n.d.]. Disable-JS extension. https://github.com/dpacassi/disable-javascript
[8] [n.d.]. How to Disable JavaScript. https://www.howtogeek.com/663569/how-to-disable-and-enable-javascript-on-google-chrome/
[9] [n.d.]. NoScript website. https://noscript.net/
[10] 2015. .
[11] 2020. Internet Overtakes Newspapers As News Outlet. https://www.pewresearch.org/politics/2008/12/23/internet-overtakes-newspapers-as-news-outlet/#summary-of-findings
[12] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 674–689. https://doi.org/10.1145/2660267.2660347
[13] Pew Research Center: Journalism amp; Media staff. 2020. News is Valued but Willingness to Pay is Low. https://www.journalism.org/2011/10/25/news-and-pay/
[14] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *2012 IEEE Symposium on Security and Privacy*. 553–567. https://doi.org/10.1109/SP.2012.44
[15] Peter Eckersley. 2010. How Unique Is Your Web Browser?. In *Privacy Enhancing Technologies*, Mikhail J. Atallah and Nicholas J. Hopper (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.
[16] Gabor Gyorgy Gulyas, Doliere Francis Some, Nataliia Bielova, and Claude Castelluccia. 2018. To Extend or Not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society* (Toronto, Canada) *(WPES'18)*. Association for Computing Machinery, New York, NY, USA, 14–27. https://doi.org/10.1145/3267323.3268959
[17] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2019. Browser Fingerprinting: A survey. arXiv:1905.01051 [cs.CR]
[18] Jonathan R Mayer. 2009. Internet Anonymity in the Age of Web 2.0. *A Senior Thesis presented to the Faculty of the Woodrow Wilson School of Public and International A airs in partial fulfillment of the requirements for the degree of Bachelor of Arts* (2009), 103.
[19] K. Mowery, Dillon Bogenreif, Scott Yilek, and H. Shacham. 2011. Fingerprinting Information in JavaScript Implementations.
[20] K. Mowery and H. Shacham. 2012. Pixel Perfect : Fingerprinting Canvas in HTML 5.
[21] Gabi Nakibly, Gilad Shelef, and Shiran Yudilevich. 2015. Hardware Fingerprinting Using HTML5. *ArXiv* abs/1503.01408 (2015).
[22] Panagiotis Papadopoulos, Peter Snyder, Dimitrios Athanasakis, and Benjamin Livshits. 2020. Keeping out the Masses: Understanding the Popularity and Implications of Internet Paywalls. arXiv:1903.01406 [cs.CY]
[23] T. Saito, K. Yasuda, T. Ishikawa, R. Hosoi, K. Takahashi, Y. Chen, and M. Zalasiński. 2016. Estimating CPU Features by Browser Fingerprinting. In *2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. 587–592. https://doi.org/10.1109/IMIS.2016.108
[24] Takamichi Saito, Koki Yasuda, Kazuhisa Tanabe, and Kazushi Takahashi. 2018. Web Browser Tampering: Inspecting CPU Features from Side-Channel Information. In *Advances on Broad-Band Wireless Computing, Communication and Applications*, Leonard Barolli, Fatos Xhafa, and Jordi Conesa (Eds.). Springer International Publishing, Cham, 392–403.
[25] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 679–694. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/sanchez-rola
[26] Elisa Shearer. 2020. Social media outpaces print newspapers in the U.S. as a news source. https://www.pewresearch.org/fact-tank/2018/12/10/social-media-outpaces-print-newspapers-in-the-u-s-as-a-news-source/
[27] Felix Simon and Lucas Grave. 2019. Pay Models for Online News in the US and Europe: 2019 Update. https://reutersinstitute.politics.ox.ac.uk/sites/default/files/2019-05/Paymodels_for_Online_News_FINAL_1.pdf
[28] Alexander Sjösten, S. Acker, and A. Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (2017).
[29] O. Starov and N. Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *2017 IEEE Symposium on Security and Privacy (SP)*. 941–956. https://doi.org/10.1109/SP.2017.18
[30] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. 728–741. https://doi.org/10.1109/SP.2018.00008