PhishPrint: Evading Phishing Detection Crawlers by Prior Profiling

ARTIFACT EVALUATED usenix ASSOCIATION PASSED

Bhupendra Acharya
UNO Cyber Center
University of New Orleans
bacharya@uno.edu

Phani Vadrevu
UNO Cyber Center
University of New Orleans
phani@cs.uno.edu

Abstract

Security companies often use web crawlers to detect phishing and other social engineering attack websites. We built a novel, scalable, low-cost framework named *PhishPrint* to enable the evaluation of such web security crawlers against multiple cloaking attacks. *PhishPrint* is unique in that it completely avoids the use of any simulated phishing sites and blocklisting measurements. Instead, it uses web pages with benign content to profile security crawlers.

We used *PhishPrint* to evaluate 23 security crawlers including highly ubiquitous services such as Google Safe Browsing and Microsoft Outlook e-mail scanners. Our 70-day evaluation found several previously unknown cloaking weaknesses across the crawler ecosystem. In particular, we show that all the crawlers' browsers are either not supporting advanced fingerprinting related web APIs (such as Canvas API) or are severely lacking in fingerprint diversity thus exposing them to new fingerprinting-based cloaking attacks.

We confirmed the practical impact of our findings by deploying 20 evasive phishing web pages that exploit the found weaknesses. 18 of the pages managed to survive indefinitely despite aggressive self-reporting of the pages to all crawlers. We confirmed the specificity of these attack vectors with 1150 volunteers as well as 467K web users. We also proposed countermeasures that all crawlers should take up in terms of both their crawling and reporting infrastructure. We have relayed the found weaknesses to all entities through an elaborate vulnerability disclosure process that resulted in some remedial actions as well as multiple vulnerability rewards.

1 Introduction

The web has been seeing an increasing amount of social engineering attacks of late. Web-based social engineering attacks such as phishing and malvertisements [45] have been on the rise [14, 15, 33]. URL Blocklisting services such as Google's Safe Browsing (GSB) and Microsoft's SmartScreen have been working as front-line defenses in protecting the users against these kinds of attacks. Most web browsers lookup domain names in these blocklists before proceeding to display the web pages to the users. For example, Chrome, Firefox, Safari, and Samsung Internet web browsers which together account for about 90% of the market share use the GSB blocklist [3]. GSB is deployed in about four billion devices worldwide and shows millions of browser warnings every day protecting users from web attacks. Such blocklists are populated with the help of web security crawlers that regularly scout web-pages to evaluate them. However, in order to evade these crawlers, miscreants employ many cloaking techniques [23, 38, 39, 49, 52].

Despite such great importance, security crawlers have been left understudied for a long time. Only recently, researchers have begun to focus on evaluating the robustness of these crawlers against various cloaking attacks [32, 37, 52]. One common design methodology that has emerged in all these works is the use of phishing experiments. This usually involves setting up multiple websites with different second-level domains (TLD+1s) as phishing sites get blocked at a TLD+1 level [5, 6, 12]. These sites are then used as unique *tokens* for hosting simulated phishing pages that are hidden by distinct candidate cloaking mechanisms. For example, some pages might deliver phishing content to only certain geo-locations or mobile user agents [37]. User interaction detectors [52] and CAPTCHAs [32] have also been utilized as cloaking mechanisms to hide mockup phishing content. The key idea in this approach is to create disparate sets of phishing token sites with different cloaking mechanisms and then selectively self-report them to different crawlers. Depending on which sets of websites get listed in the browser blocklists, one can measure which crawlers can defend how well against various cloaking attacks.

In this research, we explored an alternate approach that completely avoids simulated phishing sites and blocklisting measurements for evaluating security crawlers. Instead, we create multiple token websites with benign content and self-report them selectively to different crawlers to trigger their visits. We then directly profile the crawlers by collecting forensic information such as their IP addresses, HTTP headers and browser fingerprints at a large scale. Lastly, we conduct a quantitative analysis of this information to identify and compare multiple previously unstudied cloaking weaknesses across different crawlers. We also conduct small scale phishing experiments using phishing sites that are powered by this analysis to demonstrate the gravity of the weaknesses we found in the crawlers.

Since we avoid using phishing content in the token sites, these sites and their associated TLD+1 domains do not get blocked. As a result, we can use a single TLD+1 domain to host a large number of token profiling sites as subdomains. This allows our approach to be much more scalable than using phishing experiments. For example, in our main profiling experiment, we created and sent 18,532 token sites to 23 security crawlers at the cost of registering a single domain name. Each token site collects information about multiple cloaking weaknesses simultaneously with this design further boosting the scalability. Moreover, by using benign sites researchers can avoid the difficult process of seeking prior immunity from hosting providers as there is no risk of whole account takedown unlike in the case of phishing experiments where this is essential [37]. We discuss the question of whether this approach of using a single TLD+1 introduces bias in §6.

We implemented our approach by building *PhishPrint*, a scalable security crawler evaluation framework that is made up of two modules. The main module conducts large scale profiling experiments to help find crawler weaknesses while the other one aids in set up of small scale phishing experiments to help demonstrate the seriousness of these weaknesses. Using *PhishPrint*, we profiled 23 security crawlers over a 70-day period using 18,532 token sites resulting in 348,516 HTTP sessions. These 23 crawlers included those employed by highly ubiquitous services such as Google Safe Browsing and Microsoft Outlook e-mail scanners as well.

When building *PhishPrint*, we made use of code from a popular browser fingerprinting project [2] to help in gathering the crawlers' fingerprints along with their HTTP headers. When we analyzed the gathered data, we found several interesting crawler weaknesses. For example, we found that many crawlers carry *"Crawler Artifacts"* such as anomalous HTTP headers implying the presence of browser automation. We also saw that the Autonomous Systems (AS) used by a lot of crawlers are very uncommon (for human users) and limited in variety. Therefore, they can be embedded into an *"AS Blocklist"* to help in cloaking. Similarly, we also found that the IP addresses were limited in some cases leading to an *"IP Blocklist"*. We also found that several crawlers do not use *"Real Browsers"* as they fail to support the execution of fingerprinting code that is widely supported in the browsers of most users. Finally, we found that the entire crawler ecosystem is severely lacking in the diversity of their advanced browser fingerprints (defined here as a combination of JS-based Font, Canvas and WebGL fingerprints) thus pointing to the efficacy of a *"Fingerprint Blocklist"* to aid in cloaking.

To measure and confirm the damage that these anomalies and blocklists can do, we used them to power 20 evasive phishing sites deployed in *PhishPrint*. Despite aggressive self-reporting of all phishing sites to the crawlers, our results showed that 90% of the sites can escape blocklisting indefinitely in stark contrast to the lifetime of a baseline phishing site which was only about three hours.

We will describe *PhishPrint* in more detail in §2. All experiments and their results are presented in §3 and §4, while mitigations are covered in §5. We discuss vulnerability disclosure, limitations, ethical considerations and future work plans in §6 and related work in §7.

We make the following contributions with this paper:

1. Framework: We built a novel, scalable, low-cost framework named *PhishPrint* to enable evaluation of web security crawlers by completely avoiding the use of blocklisting measurements and phishing sites.
2. Measurements: We deployed *PhishPrint* in a 70-day longitudinal study to evaluate 23 crawlers individually and more than 80 crawlers cumulatively. Analysis of the data led us to new cloaking weaknesses and attack vectors against the crawler ecosystem (*Crawler Artifacts and Real Browser Anomalies*, *AS and Fingerprint Blocklists*) which we confirmed through user studies and phishing experiments. We also devised a simple metric named *Cloaking Vector Defense Score* to compare these weaknesses in order to help in the prioritization of fixes.
3. Impact: In order to address these weaknesses, we suggested concrete mitigation measures in areas of crawler and reporting infrastructures. We also performed a thorough vulnerability disclosure with all crawler entities eliciting their positive feedback and remedial actions.

We also state that we are willing to share our *PhishPrint* codebase selectively with all security researchers and concerned industry members to enable further evaluation studies on the security crawler ecosystem.

2 System Description

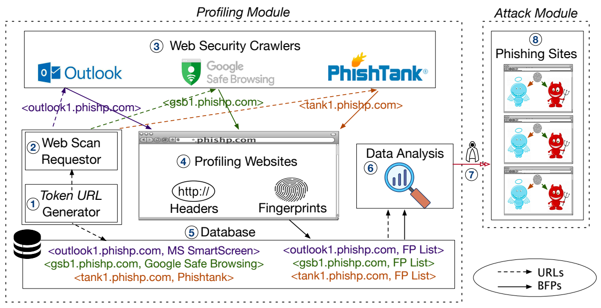

Figure 1: PhishPrint: System Overview

Our *PhishPrint* system is made up of two modules (see Fig. 1). The main module is the *Profiling Module* which uses a large number of benign websites (4) to collect and analyze sensitive profiling information from security crawlers and find any cloaking defense weaknesses. These weaknesses can then be harnessed to devise cloaking attack vectors. The efficacy of these attack vectors can then be verified with the *Attack Module* which uses an array of simulated phishing websites ⑧ for this.

The working of the *Profiling Module* begins with the *Token URL Generator* ① whose job is to periodically generate unique, never-before-seen URLs that will be given as tokens to various crawlers. The URLs are also stored in a database ⑤. Although each URL is unique, they all point to a single *Profiling Website* ④ server that we maintain. As previously discussed, we use unique subdomains of a single TLD+1 domain for generating these URLs. The mapping between the token URLs and the web server instance was set up with the help of wildcard DNS records and `.htaccess` rewrite rules.

The *Web Scan Requestor* ② receives URLs periodically from the Token URL Generator and reports them to different crawlers ③ as potential "phishing URLs". We went through an elaborate process to find a comprehensive list of security crawlers that can be supported by the requestor module. Firstly, we included crawlers such as Google Safe Browsing (GSB) and Microsoft SmartScreen which power the URL blocklists of most web browsers covering millions of users. We also added support for crawlers such as PhishTank, APWG, and ESET which along with GSB and SmartScreen have all been studied in previous research [37]. Further, we went through the list of URL scanning services hosted by VirusTotal [11] and included 17 additional crawlers that have a publicly accessible reporting interface. To our knowledge, none of these have been studied previously. We also tested various communication applications such as e-mail clients and social media apps with token URLs to see if we could find evidence of any crawlers being employed by these vendors. In this process, we discovered that Microsoft employs a crawler to pre-scan all URLs received by its Office 365 customers using the Outlook e-mail service [1]. Given that Office 365 is a hugely popular application with a current subscriber base of more than 250 million people [10], we also included it as a candidate to be evaluated bringing the total list of crawlers to 23[1] (listed in 1st column of Table 1). The Web Scan Requestor module is built to use different methods such as Selenium-based browser automation code, emails as well as direct web API calls in order to send periodic phishing URL reports to the 23 crawlers.

After receiving the reports, the crawlers will visit the token URLs. As already noted, all these URLs point to a single web server hosting a Profiling Website ④. We designed this website to be able to extract browser fingerprints of any visitor without requiring any user interaction. For this, we adapted (with permission) the fingerprinting code developed and used for the *AmIUnique* project [2, 29]. Apart from serving as a reliable cross-browser compatible codebase, their project also allowed us to be able to evaluate the specificity of some attack vectors we derive on a large scale as will be shown in later sections. The fingerprints extracted are: Canvas, JS-based Font and WebGL fingerprints (code in Appendix D) and `Navigator` object properties. The web server will store all the extracted fingerprints, the client's IP address and the HTTP request headers in the database.

After eliciting a suitable number of crawls over an extended period of time, we can ascribe the collected fingerprint data and HTTP metadata to different crawlers by using the token URLs as a common factor ⑤. We can then conduct a thorough analysis of the cloaking defenses of specific crawlers ⑥. This allows us to derive different cloaking attack vectors ⑦. The cloaking vectors can then be evaluated using the *Attack Module*. The module contains an array of evasive simulated phishing websites that are bootstrapped to use the derived cloaking vectors. It is important to note that some of these cloaking vectors rely on blocklists that need to stay up-to-date in order to remain continually effective. As a result, in order to use these kinds of cloaking vectors, the *Profiling Module* needs to continue to run throughout the run time of the *Attack Module*. More details about the setup of the phishing websites are discussed in §4.

We used PHP, Python and JavaScript for building all the above described modules with about 20K lines of code. In the interest of making a real impact in improving the security architecture of crawlers, we plan to release our code to vetted academic researchers as well as concerned members of the industry upon request. However, we are abstaining from making a public release of our code in order to deter risks of possible abuse of the system by malicious actors.

3 Profiling Security Crawlers

We setup *PhishPrint* to run on our University network. We registered a domain under the `.com` TLD for our *Profiling Websites*. As described in §2, our system was setup to collect and analyze profiling data from 23 different crawlers (Table 1). We ran the system for a period of 10 weeks beginning in the 2nd week of January 2020. We collected the data for a total period of 77 days (until the last week of March) in order to allow sufficient time for any delayed crawls that might be initiated from some crawlers. During this period, *PhishPrint* reported 12 token URLs as fake phishing reports daily to each of the 23 crawlers (Ethical considerations are discussed in §6). These reports were sent in two hour intervals of time throughout to all the crawlers. As a result, we reported a total of 840 token URLs to most crawlers[2] over the deployment period.

[1]We also discovered that some media applications such as Slack and Facebook Messenger scan our token URLs. However, we do not consider them as security crawlers as they were clearly identifying themselves with the `User-Agent` headers akin to search engine bots.

[2]Forcepoint, FortiGuard and GSB are the only exceptions. Forcepoint has a reporting limit of 5 URLs per day restricting us to 350 submitted URLs. Due to intermittent technical issues on both server and client sides, we could only report 777 and 612 URLs to FortiGuard and GSB respectively.

3.1 Analysis and Cloaking Vectors

The above mentioned setup allowed us to collect sensitive profiling data from multiple crawlers over the 10-week period. We analyzed the data to find crawler weaknesses and derive relevant cloaking vectors. The profiling data we collected for this project can be divided into these 3 categories: *browser anomalies*, *network data* and *advanced browser fingerprints*.

In this section, we will describe how the profiling data from the 3 areas was analyzed and what cloaking vectors were derived as a result. Before this, it is helpful to first establish some terminology relating to cloaking attack vectors. Regardless of the type of profiling data being used, cloaking vectors used by attackers trying to evade crawlers can fall into one of two classes: *Anomalies* and *Blocklists*. We will describe these two classes below:

Anomaly cloaking vectors capitalize on the characteristic anomalous behaviors exhibited by crawlers when visiting candidate websites. These vectors can be created after finding any anomalies in the requests being made by crawlers that strongly indicate the fact that they are not from a potential human victim. For example, consider a HTTP request made by a crawler with a headless browser's `User-Agent`. Attackers can block all such requests to avoid detection without blocking any potential victims as no victim will use a headless browser. Thus, by definition, all these vectors work with high specificity.

Blocklist cloaking vectors rely on some specific fingerprints known to be associated with crawlers (such as from *PhishPrint*'s profiling data) in order to create a blocklist for the operation of cloaking websites. For example, if there are a set of specific IP addresses that Google uses for its GSB crawlers, they can then be made part of a blocklist attack vector to evade GSB.

Blocklist vectors differ from anomaly vectors in two key aspects. Firstly, many blocklists need continuous updating in order to be effective. For example, if a crawler keeps changing its IP addresses, then the corresponding blocklists need to be updated by the attackers. This is not the case with anomaly vectors which rely on some specific crawler idiosyncrasies that are overlooked by the crawlers and hence remain fixed. Secondly, blocklists might block some potential victims. So, their specificity needs to be taken into account by attackers before using them. For example, if an attacker simply blocks all /24 subnets of IP addresses seen from a crawler and if that crawler was using a residential proxy to route its requests, then such a blocklist could potentially cause a lot of false positives for the attacker. On the other hand, anomaly vectors are all very specific as already discussed.

We will now discuss the three areas of profiling data we analyzed in our study along with the associated cloaking vectors that we derived and their novelty aspects.

3.1.1 Browser Anomalies

The first area of profiling data we consider focuses on how closely the client code used by a crawler resembles that of a real browser. We observed several anomalies in the web browsers used (or pretending to be used) by the crawlers. We categorize these into 3 different anomaly vectors and discuss them here.

JS Execution Anomaly Vector. The first anomaly we discovered was the inability of a few crawlers to execute some simple JavaScript code. For this, we checked whether or not a crawler is capable of executing a test function that is passed to `Window.setInterval()` method. This is very similar to the `onload` event-based cloaking vector used in [37] (see Appendix C for details). However, it is to be noted that many crawlers are good at executing such simple JS code and hence this serves as a baseline against which can measure the efficacy of more sophisticated cloaking vectors.

Real Browser Anomaly Vector. We designed our profiling website to ship out fingerprints (specifically: Font, Canvas and WebGL fingerprints; code in Appendix D) to the database without requiring any user interaction. We verified that this website is cross-platform compatible by manually testing it with most used web browsers such as Chrome 79, Firefox 71, Safari 11, Edge 44 and IE 11 on Windows (Vista, 7 and 10), macOS, Linux (Ubuntu), iOS and Android platforms. During this process, as and when required, polyfill Javascript libraries were used to maintain compatibility with older web browsers (such as IE) that do not fully support some APIs such as Canvas. Thus, our thorough testing ensured that the most commonly used web browsers will all ship us fingerprints as soon as they visit our website. However, we observed that many crawlers were unable to ship out their fingerprints as the fingerprinting code fails to run in their "browsers" although many of them do successfully execute the simpler JS code mentioned previously. This is highly likely due to the failure of crawler vendors in setting up robust JS-execution environments to support all advanced web APIs such as `Canvas` [20] and `WebGL` [21]. We refer to this as a *Real Browser* anomaly.

To our knowledge, no other previous research has attempted to do such analysis against security crawlers[3].

Crawler Artifacts Anomaly Vector. We discovered a few anomalies when manually analyzing the HTTP requests headers and `navigator` objects we collected from the crawlers. For some crawlers, we saw that the `navigator.useragent` value does not match the `User-Agent` header. Similarly, `navigator.platform` does not always match the platform indicated in the `User-Agent` header. For example, it was common to see cases where the `User-Agent` header indicates a Windows platform, but the `navigator.platform` indicates a Linux platform. Similarly, we saw a number of cases where the `User-Agent` bears *indicators of automation* such as `curl`, `phantomjs`, `headless` etc. Further, we also found discrepancies in the values of `navigator.webdriver`. This is a Boolean field that indicates whether a web browser is being driven by browser automation software such as Selenium. While for most web browsers the default value of this field in

[3]The "Real Browser" vector described in [37] is synonymous with the JS Execution Anomaly we discussed.

a non-automated browser is set to `false`, in Chrome it is set to `undefined`. In this regard, we noticed that in some crawlers, `navigator.webdriver` was being set to `false` even though the `User-Agent` indicated a Chrome browser. This is a clear anomaly and shows that the property had been tampered with.

We note that previous works have used similar techniques in a more elaborate fashion to defeat privacy-protecting browsers and extensions [46] and ad network blockages [45]. [26] also found several such artifacts by studying in-the-wild bot detection scripts. However, in our study, we measure these anomalies as weaknesses of security crawlers and apply them for cloaking.

3.1.2 Network Data

For this part of the analysis, we focused on the IP addresses used by the crawlers for initiating web requests to *PhishPrint*. We collected these addresses during our deployment period and crafted **IP Blocklist Vectors**. Thus, we were able to mine a blocklist cloaking vector from *PhishPrint*'s data. Note that real-world attackers tend to use massive blocklists made of IP addresses for building phishing sites [38]. Hence, it is very important to measure how well crawlers are doing (both specifically as well as cumulatively) in defending against this vector. The performance of crawlers against *in-the-wild* IP blocklists has been studied before [37]. However, the highly scalable nature of *PhishPrint* now allows us to directly collect a large amount of network infrastructure data and then analyze and compare this across an extensive set of security crawlers. In addition, we also mapped the collected IP addresses to their associated countries in order to measure the geolocation variety of the network infrastructure setup by the crawlers.

AS Blocklist Vector. Upon analyzing the Autonomous System (AS) names of the collected IP addresses, we also discovered that many crawlers are housing their crawlers in IP address spaces that can be mapped to web or cloud hosting companies (such as Amazon, DigitalOcean) or the organizations related to the crawlers themselves (such as Google, Microsoft, BitDefender, Cisco). We were able to make a list of 66 such AS names. We refer to this as an *AS Blocklist*. As it is unlikely for a potential victim to be visiting an attacker's website from such IP addresses, an attacker can easily use as AS Blocklist to evade crawlers.

AS Blocklist is a hybrid between anomaly and blocklist cloaking vectors. Similar to anomaly vectors, it is based on an anomaly and is relatively static as it is unlikely for offending crawlers to frequently change their network infrastructure between different cloud networks. On the other hand, similar to other blocklist vectors, it takes extensive data collection efforts to construct lists like this as there a myriad number of web hosting entities. Further, if the blocklist is poorly constructed and includes AS names of victim IP spaces, then there could be specificity issues as with other blocklist vectors. We empirically demonstrate that this is not the case with *AS Blocklist* with a large-scale user study later (§4.2).

Prior works have utilized AS level features to escape malware sandboxes [51]. In this study, we applied similar techniques to study security crawler evasion.

3.1.3 Advanced Browser Fingerprints

Recent privacy-oriented studies such as [17, 22] have shown Canvas, WebGL and Font list (obtained via JS) fingerprinting to be among the most discriminatory identifiers. As a further testament to this, these browser fingerprints have also been used to develop authentication schemes [13, 27]. At the same time, privacy researchers have also shown that such fingerprints are not easy to defend against and require elaborate measures [24, 31, 41, 50]. Given this, there is a high potential for developing an effective cloaking vector if crawlers do not take adequate measures to defend against these fingerprinting techniques. Hence, we wanted to analyze these fingerprints after we collected them from crawlers. Snippets of the fingerprinting code we use are listed in Appendix D. For both Canvas and WebGL fingerprints (both first introduced in [34] and later used in [22]), the code draws a hidden image on the webpage and a cryptographic hash of that image is produced to be used as a fingerprint. For font fingerprinting, a simple trick first proposed in [36] and later used in [22] is used to detect the list of 1043 fonts that are installed in the client using JavaScript. A cryptographic hash of the font list serves as the font fingerprint for the client.

Our analysis showed that the entire crawler ecosystem exhibits very little dynamism across these three fingerprints. To capitalize on this, we propose a blocklist cloaking vector. For this, we follow the approaches of prior studies [22, 29] and use a tuple of the three fingerprints: `<Font, Canvas, WebGL>` (or `<F,C,W>`) in order to effectively combine their individual fingerprinting capabilities. In the rest of this paper, we refer to this compound fingerprint as "fingerprint" for brevity. Our proposed **`<F,C,W>` Fingerprint Blocklist Vector** for this simply stores all `<F,C,W>`s seen from crawlers in the past to aid future evasion. As this is a blocklist vector, we will perform multiple measurements to verify its specificity.

3.2 Profiling Analysis Results

In this section, along with an overview of the profiling data we collected during the 10-week study, we will present measurements indicating the performance of the crawlers against the six cloaking vectors we introduced previously. All these results are presented in Table 1 where the 1st column lists all the crawlers we studied.

VT Sharing. During analysis and investigation, we found that 8 crawlers have shared their token URLs with VirusTotal [11] (VT). This sharing has taken place at varying degrees. Malwares and Quttera have shared more than 99.5% of their URLs with VT, while Bitdefender and PhishTank have shared about 10 and 30% of their URLs with VT. VT hosts more than 80 crawlers that begin scanning the uploaded URLs almost immediately. As a result, all such VT-shared URLs need to be considered separately. For this, we created a "virtual crawler" named "VT Ecosystem" and consider all VT-shared URLs exclusively here. We treat this virtual crawler as equivalent to other crawlers in the rest of this paper. Since Malwares and Quttera shared most of their URLs with VT, no meaningful specific analysis can be made for these crawlers. Hence, we avoid their individual rows in the table and just show them as part of the VT ecosystem. It is to be noted that due to the large number (80) of crawlers hosted on VT (including 18 of our 23 crawlers), the VT ecosystem can be considered as a cumulative representative of the entire crawler ecosystem.

The 2nd column shows the number of URLs submitted (discussed in §3.1), the number of URLs scanned by the crawlers and the number of URLs shared with VT by each crawler. Overall, in the 10 week period, we submitted about 18,532 *token* URLs (with distinct domain names) to all the 23 crawlers. In terms of crawl back rates, most of the crawlers did well with many of them visiting more than 90% of the submitted URLs. A notable exception is Norton which visited only 53 of the submitted URLs. The total number of URLs submitted to VT by other crawlers was 803. The 3rd column describes the number of URLs remaining to be analyzed after we excluded the VT URLs. It also lists the number of sessions established for the analyzed URLs indicating the total number of visits made. While crawlers from PhishTank and Scumware establish 50 to 100 sessions for each analyzed URL, some others such as GSB, SmartScreen and Forcepoint visit each URL only once or twice. Overall, as many as 348,516 sessions were established for scanning 18,532 distinct URLs we submitted. The 4th column shows the median of time deltas between the first crawl time and the URL submission time for URLs submitted to each crawler. Some crawlers such as Fortinet and SmartScreen have a slow average response time whereas many others including GSB, Outlook take only a few seconds.

CVD Scores. In order to compare the performance of all the crawlers across the six cloaking vectors, we need an intuitive performance metric. For this, we devised a simple metric called *Cloaking Vector Defense Score (CVD score)*. The CVD score can be computed for any given crawler (say, W) and a cloaking vector (say, V). Assume that we reported x URLs to W and it scanned y of them (ignoring the VT-shared URLs) during our entire study. We conduct an *a posteriori analysis* of all the y URLs to determine how many of them were visited *at least once* by a crawler that *does not* exhibit the weakness associated with V. If such a number is z, we report the CVD score of the pair (W,V) as $\frac{z}{y} \times 100$.

Doing this a posteriori analysis for an anomaly vector is straightforward as we simply need to determine *if at least one* of the many requests a URL might receive does not exhibit the anomaly being considered. However, in the case of blocklist vectors, we will need the respective blocklists in order to make this determination for a given request. We build this blocklist dynamically using all the historic data collected from the crawler prior to the current request. For example, in order to determine if a request r at time t can be blocked by a blocklist vector V, we use all prior requests to the crawler before t to build a blocklist and see if the current request can be blocked by such as blocklist. If it does, we determine this to be a *weak* request and do not consider it.

From the above, we can see that the CVD score, by definition, reflects *the chance (as a %) of a given crawler to successfully defend against a given cloaking vector.* Columns 5 to 10 show the CVD scores of the crawlers over the six vectors we described previously. We use red, yellow and green colors in the table to show the bad (<33), moderate($33-66$) and good (>66) scores respectively.

Anomalies. Column 5 shows that the CVD scores for JS Execution Anomaly vector are good all across the spectrum of crawlers. This demonstrates a positive evolution from the situation in [37] which showed that only 1 of the 5 studied crawlers had good score. More such evolutionary changes in crawlers have been described in Appendix C. On the other hand, many crawlers seem to be failing in handling the Real Browser Anomaly vector that we developed (Column 6). The only notable exceptions to this are APWG and the VT ecosystem. We noticed that GSB, for example, completely fails to support the WebGL API in many of its crawlers. Some other notable failures are Outlook, Avira and Forcepoint that did not visit even a single submitted URL with a Real Browser. The overall combined CVD score of all crawlers in this respect is thus only 35.2 which shows a lot of scope for improvement. A positive result is that most vendors seem to have some crawlers that do not carry Crawler Artifacts Anomalies (Column 7). However, all crawlers from AlienVault and Avira have an anomalous `navigator.webdriver` property which was causing all their visits to be easily evadable.

IP-Blocklist. The CVD scores for IP-Blocklist vector along with the number of distinct IP addresses of source requests and the countries they are associated (# CCs) with is in Column 8. We note that as many as 11 crawlers make their visits from less than 20 distinct IP addresses even though they visit hundreds of domains forming thousands of sessions across time. Crawlers from AlienVault and OpenPhish visit only from 1 or 2 IP addresses. A control experiment reported that this situation persists even when doing repeated reports from diverse sources (§3.2.2). On the other hand, URLs submitted to some crawlers including GSB, Outlook, PhishTank and APWG are scanned by a large number of distinct IP addresses. For PhishTank, this number is as high as 4096 IP addresses (spread over 51 countries) for the 579 URLs we analyzed. Figure 2 charts the growth of the distinct number of IP addresses we have seen across the days of our experiment. The graphs shows a near-linear growth for APWG and GSB indicating the greatest diversity in IP addresses. SmartScreen shows an interesting IP infrastructure growth. The number of IP addresses was 1 for the first 50 days of the experiment but has risen to 50 in the last 20 days. This indicates an infrastructure change during the last 20 days which was referred to during our vulnerability disclosure process as well. One more interesting point to note is the number of countries

① Crawlers	② # URLs Submitted / Scanned / VT Shared	③ # URLs Analyzed / # Sessions	④ Reply Time h:m:s	Browser Anomalies			Network Data			Advanced BFPs
				⑤	⑥	⑦	⑧		⑨	⑩
				JSE-A Score	RB-A Score	CA-A Score	# IPs / # CCs	IP-B Score	AS-B Score	# <F,C,W>s / #F - #C - #W (FCW-B Score)
AlienVault	840 / 837 / 0	837 / 2354	0:00:16	99.5	18.9	0	1 / 1	0.1	0	2 / 1-2-2 (0.2)
APWG	840 / 839 / 0	839 / 4658	0:00:10	100	99.5	99.8	2726 / 8	99.1	62.9	6 / 7-7-3 (0.6)
Avira	840 / 837 / 0	837 / 2082	0:50:27	92.1	0	0	70 / 3	8.4	43.0	0 / 0-0-0 (0)
Badware	840 / 837 / 0	837 / 837	0:00:08	99.8	0	100	1 / 1	0.1	100	0 / 0-0-0 (0)
Bitdefender	840 / 542 / 67	475 / 3918	4:16:10	97.9	40.2	97.3	62 / 10	9.1	79.6	46 / 46-38-12 (9.3)
Dr.Web	840 / 836 / 0	836 / 846	0:00:22	79.8	0	0	15 / 3	1.8	71.8	0 / 0-0-0 (0)
ESET	840 / 764 / 0	764 / 987	3:35:02	99.7	17.9	100	12 / 2	1.4	99.9	6 / 3-6-3 (0.8)
Forcepoint	350 / 295 / 0	295 / 295	0:00:24	85.1	0	45.8	1 / 1	0.3	100	0 / 0-0-0 (0)
FortiGuard	777 / 764 / 8	756 / 4590	0:00:46	97.1	9.4	100	19 / 3	2.0	12.7	27 / 25-25-8 (3.4)
Fortinet	840 / 772 / 5	767 / 4495	11:45:36	98.8	5.9	100	2 / 2	0.3	7.4	12 / 12-11-6 (1.6)
GSB	612 / 591 / 0	591 / 775	0:00:04	99.2	23.9	100	619 / 83	94.4	90.9	2 / 2-2-2 (0.3)
SmartScreen	840 / 822 / 0	822 / 1133	2:58:11	99.8	44.0	77.6	50 / 2	2.6	100	17 / 13-8-5 (1.7)
Norton	840 / 53 / 0	53 / 69	0:31:42	86.8	13.2	88.7	19 / 3	34.0	98.1	1 / 1-1-1 (1.9)
Notmining	840 / 838 / 0	838 / 1675	0:00:10	84.3	0	0	1 / 1	0.1	0	0 / 0-0-0 (0)
OpenPhish	840 / 835 / 0	835 / 4928	1:00:02	99.8	59.6	100	2 / 2	0.1	0	1 / 1-1-1 (0.1)
Outlook	840 / 672 / 0	672 / 676	0:00:18	98.7	0	100	535 / 1	79.5	0	0 / 1-1-0 (0)
PhishTank	840 / 838 / 259	579 / 45976	0:00:10	100	82.2	100	4096 / 50	93.4	100	51 / 55-69-19 (7.4)
Scumware	840 / 633 / 2	631 / 29537	0:25:47	100	80.0	100	1643 / 59	82.9	100	27 / 37-32-5 (3.0)
Sophos	840 / 793 / 0	793 / 2170	0:01:47	97.6	3.5	91.2	26 / 3	2.0	100	3 / 2-3-1 (0.4)
Sucuri	840 / 830 / 0	830 / 2488	0:00:09	87.2	0	100	837 / 70	100	96.6	0 / 0-0-0 (0)
ZeroCERT	840 / 840 / 462	378 / 1152	0:05:11	100	0.5	100	3 / 1	0.8	100	1 / 2-2-1 (0.3)
VT Ecosystem	2483 / 2465 / -	2465 / 232875	0:04:18	99.9	98.8	100	7795 /76	82.1	99.8	101 / 111-97-21 (3.1)
All	18532 / 16730 / 803	16730 / 348516	0:01:15	96.3	35.2	77.4	15394 /113	33.4	65.6	204 / 182-162-36 (1.1)
Best Score	-	-	-	100	99.5	100	-	99.1	100	9.3

Table 1: Details of 70-day profiling study including CVD scores for the six cloaking vectors
The scores are color-coded: red for <33, yellow for $33-66$ and green for >66.
JSE-A: JS Execution Anomaly; **RB-A:** Real Browser Anomaly; **CA-A:** Crawler Artifacts Anomaly;
#IPs: Crawler IP Addresses; **#CCs:** Country Codes; **IP-B:** IP Blocklist; **AS-B:** AS Blocklist;
#<F,C,W>s: Font/Canvas/WebGL fingerprint tuples; **#F:** Font fingerprints; **#C:** Canvas fingerprints;
#W: WebGL fingerprints; **FCW-B:** <F,C,W> Fingerprint Blocklist;

associated with the IP addresses. APWG is an interesting example, in that even though they employ 2726 IP addresses, they are all associated with only 8 countries which makes a country-based cloaking vector feasible for targeting victims outside those 8 countries. The CVD scores demonstrate a very polarized situation with roughly half the crawlers having very good scores >80 and half having very bad scores <10.

AS-Blocklist. Many crawlers including Outlook and AlienVault showed bad AS-Blocklist CVD scores. Outlook in particular, was using crawlers that were all housed in a Microsoft IP space and is hence evadable despite using a large number of IPs for visiting the URLs. The same is the case with FortiGuard, Avira and OpenPhish who were using common cloud and web hosting companies for housing their crawler bots. On the other hand, there were several crawlers such as PhishTank and GSB that performed well in this respect.

<F,C,W> Fingerprint Blocklist. Column 10 shows the the number of distinct <F,C,W> fingerprints and the individual Font, Canvas and WebGL fingerprints collected from the crawlers. It also shows the <F,C,W> Fingerprint Blocklist CVD scores considering the fingerprint tuple. Despite scanning 16,730 distinct domains and initiating 348,516 HTTP sessions over 70 days, we see that the crawlers collectively only had 204 distinct <F,C,W> fingerprints including 162 Canvas and 182 Font fingerprints. These numbers can be put into perspective by seeing that a prior study [22] has collected as many as 78K distinct Canvas FPs and 17K distinct Font FPs over a 6-month period with the help of a few regional websites[4]. Further, we can also notice these crawlers used as many as 15,394 distinct IP addresses in total. This shows that while many vendors are actively trying to change their network infrastructure fingerprints, little is being done to vary the advanced browser fingerprints.

Inspecting the individual rows, we can see that even vendors that invested a lot into their network infrastructure such as GSB and APWG only have a handful of distinct <F,C,W>s (2 and 6). Note that 7 crawlers have a 0 score in combating the Real Browser Anomaly cloaking vector. This means their browsers are not even capable of running the fingerprinting code and hence we did not collect any <F,C,W>s from them. Some crawlers such as PhishTank, Bitdefender and the VT ecosystem fare slightly better with 51, 46 and 101 distinct

[4]This study did not include *AmIUnique*'s current WebGL FP implementation. Further, our experiments showed they are the least specific of the 3 fingerprints §3.2.1. Hence, we avoid discussing WebGL fingerprints here. However, we do use these as part of the <F,C,W> tuple as already described.

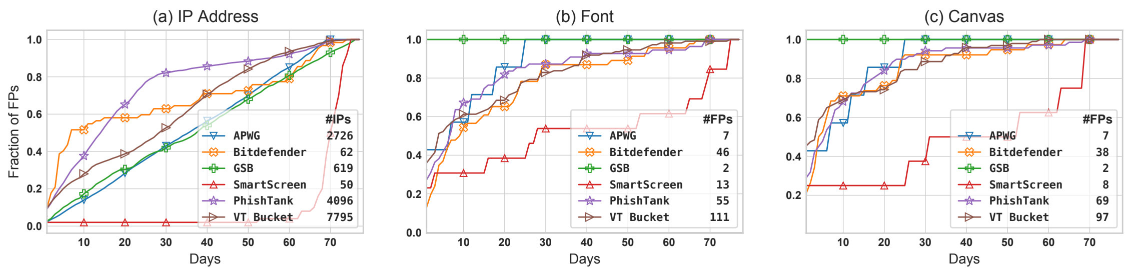

Figure 2: Growth of distinct IP addresses and fingerprints in *PhishPrint database* for different crawlers

values of <F,C,W>s. However, these still not seem to be sufficient as demonstrated by their fingerprint CVD scores. Bitdefender's score although the highest among all crawlers is still only 9.3. This means that more than 90% of the token URLs would have evaded detection from Bitdefender using the <F,C,W>s as a simple blocklist. It is also interesting to see the growth rate graphs of distinct Font, Canvas fingerprints collected by *PhishPrint*. While the IP addresses had an almost linear growth for many crawlers (such as GSB and APWG), the Font and Canvas fingerprint growth rates present a completely opposite picture. As GSB has only 2 such fingerprints that were used from day 1, the graph is just a flat line. For APWG, PhishTank and Bitdefender, the growth rate is very low in the last 30 days. This indicates the high likelihood of a successful blocklist cloaking vector which we will demonstrate later §4. SmartScreen has only 17 <F,C,W>s for its 822 URLs. The growth rate for these is in a step-wise fashion with long flat lines indicating again the utility of a blocklist cloaking vector.

Further, this best score of 9.3 remains in very stark contrast with best scores for the other five cloaking vectors as shown in the final row of the table. This shows that while the other cloaking vectors are being well handled by at least some crawlers, *advanced fingerprints such as <F, C, W>s present a grave cloaking weakness that seems to be affecting all the entities in the crawler ecosystem.*

3.2.1 Specificity of Advanced Fingerprints

As we are proposing to use <F,C,W>s as a blocklist for evasion, their specificity needs to be established as already discussed. We accomplished this by collecting a set of <F,C,W>s from crawlers and measuring how common they are among internet users. For this, we re-deployed *PhishPrint* on 3 days spread evenly over September 2020. We collected all <F,C,W>s from 5 crawlers (listed in Fig. 3)[5] by sending 12 token URLs each day to each of the 5 crawlers. It is to be noted that 35 of these 180 URLs (including 34 PhishTank URLs) were shared with VirusTotal immediately, thereby soliciting crawls from many of the 80 VT crawlers similar to the longitudinal study. At the end of each day, we waited for a 24 hour period and then sent the collected fingerprints to AmIUnique project's API [2]. This allowed us to directly measure the percentage of AmIUnique visitors who have the same Font, Canvas or WebGL fingerprints. Each time we made a query, the API would look up the data of visitors in the past 90 days[6]. As our 3 querying days are spread across a one-month period, the datasets of visitors against which our fingerprints were compared on each day is different. Among the 3 datasets, the smallest dataset is made up of 467,696 visitors.

Table 2 summarizes the results of all our queries. In total, we collected and queried 53 Font, 46 Canvas and 16 WebGL fingerprints. Interestingly, we noticed that many crawlers continued to carry the same BFPs as the ones we saw in our longitudinal study despite the 6-month difference in time. For example, all fingerprints collected from APWG and

BFP	#	Unique	Median	75%	95%	Max	Sum
Font	53	20	0.0009%	0.042%	2.16%	12.46%	25.46%
Canvas	46	11	0.0034%	0.07%	1.57%	2.17%	10.47%
WebGL	16	1	0.081%	2.09%	5.53%	11.47%	25.63%

Table 2: Specificity of Crawler Fingerprints

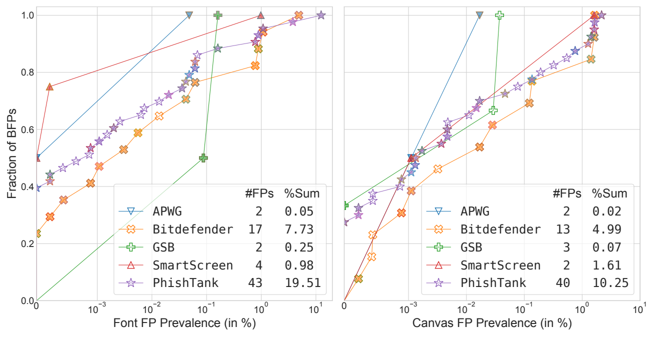

Figure 3: CDF plots showing the prevalence (in %) of crawler fingerprints among 467K web users. Solid markers indicate fingerprints that were also found in the 10-week study[7]

[5]We chose these 5 crawlers based on fingerprint diversity and popularity. We limited to only these as our prior agreement with AmIUnique project developers (our data source) limited us to only 100 fingerprint look-ups. We will show in §4.2 that these crawlers are mainly responsible for most fingerprint collisions with users.

[6]Canvas and Font fingerprints are relatively stable for a user with a median lifetime of more than 9 months [47].

[7]For some crawlers such as PhishTank and Bitdefender the markers on the y-axis refer to multiple fingerprints. In these cases, if at least one of those fingerprints was found in the previous study, we marked that point solid.

Config	# Sessions	#IPs	# <F,C,W>
SR	954	291	10
DRR	799	188	10
Similarity	-	0.046	0.54

Table 3: Results of Diverse Repeated Reporting Experiment

SmartScreen were already seen in the previous study. In total, 71.3% of fingerprints were already collected previously. The table shows the distribution of prevalence (in %) of the 3 fingerprints and the graphs in Fig. 3 break this data down by each crawler. Similar to results from prior privacy-oriented studies [29], this data shows that most of the fingerprints are very rare with only a handful of them being prevalent in more than 1% of the visitors. For example, as many as 20 Font fingerprints were unique and not seen among any of the visitors. The table also shows the sum of all these prevalence percentages which could be used as a direct measure of specificity if these fingerprints were individually used as cloaking vectors. For example, the lowest of these is 10.47% for Canvas thus indicating that attackers will only lose about 10.47% of potential victims if they were to use a blocklist made solely of Canvas-based crawler fingerprints as their cloaking vector. However, if they used a more specific cloaking vector such as the triplet `<F,C,W>`, we could expect this lost victim percentage be even lesser. Fig. 3 shows that this situation is even bleaker for individual crawlers with both APWG and GSB's Canvas fingerprints accounting for only 0.02% and 0.07% of all visitors. Thus, despite their massive network infrastructure, due to this extreme lack in the diversity of fingerprints, attackers can specifically evade these crawlers without fear of losing any potential victims.

3.2.2 Diverse Repeated Reporting Experiment

In August 2020, we performed another control experiment. The goal of this was to study any potential effects that repeated reporting of token URLs from diverse sources could have on the profiling information that is collected from the crawlers. We note that during our 10-week longitudinal study, we only registered a single domain and reported each token site's subdomain created under it only a single time to each crawler. To replicate this setup, we created a similar configuration in this study by creating 50 different subdomains under a single `.xyz` domain (called `SR`). We then set up an alternate configuration for diverse and repeated reporting (called `DRR`) by using 5 different `.xyz` TLDs. On a single day, we used *PhishPrint* to report each of the 50 `SR` URLs to 10 different crawlers from a U.S. IP address (namely: AlienVault, APWG, Fortinet, GSB, SmartScreen, Norton, OpenPhish, Outlook, Sophos and ZeroCert). On the same day, we used a private VPN provider to connect to 10 IP addresses located in 7 different countries around the world and submitted 10 reports on each domain in `DRR` set to all the 10 crawlers. For some crawlers such as Outlook and APWG which use e-mail reporting, we created and used 10 different e-mail addresses for each domain in the `DRR` set. This setup ensured that an equal number of URL reports get sent from each configuration (50 to each crawler) in order to keep the comparison balanced.

Table 3 shows an overview of the comparison between the profiling results obtained from the two configurations. We can see that despite repeated diverse reporting, the URLs reported via the `SR` configuration involved more sessions and more crawler IP addresses. We surmise that this could be a result of some crawlers deliberately ignoring repeated reports even if coming from diverse sources. For example, we noticed that PhishTank's website shows an error message saying the URL is already submitted even if we try to do a repeat submission from a different account. The final rows show the Jaccard Similarity between the IP addresses and the `<F,C,W>` sets from the two configurations. The high variability in the IP addresses used by crawlers could have caused the low similarity between the two sets. It is to be noted that if we consider the associated Autonomous Systems of the IPs instead the similarity rises to about 0.57. Furthermore, breaking these results down by crawlers shows that all the ASs used by 8 of the 10 crawlers in the `DRR` set are also present in the `SR` set. The same is the case with `<F,C,W>`s where 4 of the 6 crawlers from which we collected `<F,C,W>`s have a complete match. Further, the crawlers such as OpenPhish and AlienVault which showed very little diversity in their IP space previously, have displayed exactly the same behavior in the `DRR` set as well. Overall, this experiment shows that using a non-diverse URL reporting setup as we did in our study can still enable the collection of valuable profiling information.

4 Evading Security Crawlers

Our analysis of the profiling data from the crawlers showed that five of the six cloaking vectors we devised can exploit existing weaknesses in crawlers: Real Browser Anomaly, Crawler Artifacts Anomaly, IP, AS and `<F,C,W>` fingerprint Blocklists. In this section, we present supplementary experiments that directly put these five vectors in action against crawlers as well as real users with goals to directly assess and confirm their evadability benefits as well as false positive consequences for the attackers.

4.1 Phishing Experiments

In order to confirm the real-world utility of these vectors, we build phishing websites powered with the candidate cloaking vectors, self-report them to crawlers and measure how long they will survive without getting blocked in any of the browsers (via a *Monitoring Module*). Here, our approach will be similar to prior works [37, 39]. These experiments constitute the *Attack Module* (8) depicted in Fig. 1.

4.1.1 Setup

For these experiments, we built two kinds of phishing websites: ***Baseline sites*** which do not employ any cloaking and ***PhishPrint-cloaked sites*** which use all the five cloaking vectors. The cloaked sites show benign content if any of the 5 vectors decide that the visitor should be given a cloaked page. All the cloaking logic is implemented in server-side using a simple PHP script. The phishing payload stays encrypted (using AES-256) until the cloaking logic returns a key to the client. If any of the five vectors decide not to show the phishing content, then some benign content gets loaded into the pages instead of the phishing payload. We have used two kinds of simulated phishing payloads: "PayPal" and "Bank of America". For benign payloads, we built multiple simple web pages discussing topics such as food and famous personalities.

As mentioned before in §2, the *Profiling Module* needs to be running in parallel during these experiments in order to keep the IP and fingerprint blocklists updated with the latest data. Hence, we have started performing this experiment 25 days after *Profiling Module*'s deployment (first week of February 2020) to give some bootstrap time for the two blocklists to be populated while still allowing both modules to run in parallel. In the rest of this section, we will discuss the site monitoring, reporting and web site configuration aspects of these experiments.

Monitoring Module. As the ultimate goal of the attacker is to be able to continue to deliver phishing content to the victims, we built a fully automated *Monitoring Module* to periodically check if phishing sites are still functional or blocked in web browsers. We have chosen Chrome, IE and Opera desktop browsers for this as they employ different blocklists (GSB, SmartScreen and Opera) that cover most of the web users [37, 39]. The module loads phishing sites inside the browsers and checks if the sites are blocked or not. As browser automation libraries tend to disable browser blocklists [37], we used a web-based cross-browser testing platform [4] for this purpose. The monitoring module runs a headless Chrome browser to open the target site in the testing platform and uses an OCR library to do the liveness check. We found this to be a more light-weight approach than using VMs as suggested in [37]. We did this check every 2 hours for each site. Note that while previous studies have done this monitoring for 3 [37] or 7 [39] days, we kept monitoring our sites for a period of 1 month in order to capture any late blocking that might happen due to our aggressive reporting strategy.

Aggressive Reporting. We used our Profiling Module's *Web Scan Requestor* (② in Fig. 1), to self-report all our website to crawlers. In contrast to prior works that reported each phishing site only one time [37,39] to a few crawlers, we opted for a much more aggressive approach where we repeatedly report each site (once daily) over a period of two weeks to all the 23 crawlers. It is to be noted that during the longitudinal study, we noticed that a couple of crawlers share most of their URLs to VirusTotal. Confirming similar behavior, we found that all our phishing sites have been shared with VirusTotal too. As a result, our phishing sites were shared and scanned by more than 80 crawlers that are hosted on VirusTotal as well.

Site configuration. For our experiments, we set up a total of 26 phishing websites. All 26 websites were hosted with accounts by the same hosting provider (Hostinger) and had different domain names. We used 6 of these as baseline sites (with 3 free accounts) and 20 of these as the cloaked sites powered by the 5 cloaking vectors (with 1 paid account). It is to be noted here that despite multiple requests and conversations about the nature of our research, we were unsuccessful in getting immunity for any of our accounts from the hosting provider. We chose only 6 sites for our baseline as there are already prior studies [39] establishing clearly the baseline blocklisting speed. For the same reason, we did not choose to register separate domains for these 6 baseline sites but used the free subdomains (TLD+2 level) provided by the hosting provider to conserve financial resources. For the cloaked sites, we registered 20 different `.xyz` domain names as we were unable to obtain that many free subdomains. Other than this minor difference, the setup for the experiments for both sets of sites is exactly the same. In order to prevent pre-emptive blocklisting of our websites without scanning [37], we avoided deceptive keywords such as 'paypal' or 'bank' in the URLs for the phishing pages. We instead used benign content related words for all the URLs.

4.1.2 Results

The results show that our 6 baseline sites were quickly blocked on all the browsers. Chrome (GSB) was the quickest to do this in 3 hours and 10 minutes. In fact, all the browsers blocked the 6 sites in about 10.5 hours. This agrees well with a recent large-scale study done on browser blocklists [39] which showed that the fastest blocklist (GSB) would block most of its 324 baseline sites in about 3 hours time. On the other hand, none of the 20 *PhishPrint*-cloaked sites were blocked in the first four days despite repeated reporting to all the crawlers. In the one-month period in which we did the monitoring, only 2 sites (say, 'A' and 'B') got blocked as shown in Table 4. A was blocked on day 5[8] while B got blocked on day 16. It is to be noted that even for cloaked sites such lengthy blocking time is highly unusual. For reference, [39] showed that most cloaked sites either get blocked in a few hours or remain unblocked. Given this, we surmised that both the blocked sites were due to manual vetting. To confirm this, we investigated site A's case by reloading the site in the browser that first blocked it (Opera). We noted that the browser message specified the source for the blockage as a third-party report from PhishTank. When we looked up the URL on PhishTank, we saw that our site was manually verified as a phishing URL by 3 users thus confirming our suspicions. Interestingly, we note that 1 user has also marked our site as a benign site.

As for site B, we found that it was not blocklisted by any browser, but was taken down by `xyz` registrar on day 16 due to an abuse report. We were unable to get further details on what the source for this report could be. The remaining 18 cloaked sites continued to be functional throughout the

[8]Our cloaked sites experienced a 10 hour down time after A got blocked as our hosting provider disabled our account. We then moved all our sites to another provider (Namecheap).

Type	# Sites	Alive Time
Baseline	6	3h, 10min
Cloaked site A	1	4 days, 11h
Cloaked site B	1	15 days, 14h

Table 4: Lifetimes of the blocked phishing sites

Source	# Users	# Distinct	# Unique	# Shared	Normalized Entropy
`<F,C,W>` - Ours	1007	592	469	123	0.865
`Canvas` - [29]	118,934	8,375	5,533	2,842	0.491
`Canvas` - [22]	2,067,942	78,037	65,787	12,250	0.407

Table 5: Analysis of fingerprints from user studies

monitoring period of 1 month. We verified manually that even at the time of writing this manuscript in September 2020, the 18 remaining phishing sites are still live and loading the phishing payloads on all the major browsers. Thus, we can conclude that the five cloaking vectors powered by *PhishPrint* are very effective in vastly increasing the survival chances and lifetimes of phishing websites.

4.2 User Study Experiment

Along with evasive power, we also need to study the specificity of these vectors and confirm that they are not excluding a lot of potential victims. For this, we did an empirical evaluation with the help of a user study. We modeled our experiment as a survey on the MTurk platform as this allowed us to ensure that unique workers take part in our experiment. We designed our experiment such that after obtaining prior user consent, users are exposed to a web page with exactly the same client-side fingerprinting code and server-side cloaking logic as in the phishing experiments. However, we removed the phishing payloads for this experiment to avoid showing malicious content to real users. Also, same as in the phishing experiments, the IP and `<F,C,W>` blocklists were powered by the data collected by the profiling module in real-time. In the end, we made measurements of whether or not any of the cloaking vectors decide to show cloaked content to the visiting users.

We received an exemption from our university IRB board for this experiment. In compliance with the terms of the exemption, we took measures to not store any sensitive information persistently such as the `<FCW>`s or any other information identifying the users such as IP addresses and request headers in our web servers. But, we did store the AS information for each client's IP address in order to gauge the geographical variety in locations of participants.

We performed this experiment from the third week of February 2020 to the first week of March 2020 as the profiling module was collecting data for the longitudinal study. 1150 unique users participated in our study that lasted about 16 days. 66% of the participants in our study were from the United States. However, the remaining 34% of participants were spread across 35 countries in 6 continents. Overall, the results showed that *PhishPrint*-powered cloaking logic decided to show phishing content for 79% of the users. These numbers are 76.1% for U.S. users and 81.4% for non-U.S. users. This shows that the *PhishPrint*-based evasive cloaking logic is largely specific to crawlers and can inflict harm on a large portion of users irrespective of their geographical location.

False Positive Analysis. Breaking down the 21% false positive rate by cloaking vectors, we saw these numbers: `<F,C,W>` Fingerprint Blocklist: 17.5%, AS Blocklist: 1.7%, Crawler Artifacts: 1%, Real Browser: 0.7%, IP Blocklist: 0.1%. This was expected as other than fingerprints, all others cloaking vectors are known to be specific either by definition (anomaly vectors and ASs) or due to the nature of the identifier used (chance of a crawler and a victim sharing the same exact IP address is very low).

In order to understand the reasons for the overlap between `<F,C,W>` fingerprints of MTurk users and crawlers, we sought a more permissive IRB exemption allowing us to store user fingerprints. We then performed a second MTurk study during a 10-day period in January 2021 with 1007 participants in which we collected 592 distinct `<F,C,W>` fingerprints. Table 5 shows more details of these fingerprints. While 469 of the collected fingerprints are unique, there are also 123 "shared" fingerprints each of which belong to at least 2 users. The two most popular of these shared fingerprints were only among 23 and 20 users. Further, more than 25% of them are shared among at least 5 users and more than 55% of them are shared among at least 3 users each. This shows that while there are a large number of unique fingerprints, there also exist many shared fingerprints with no small subset of fingerprints being extremely dominant. Interestingly, prior fingerprinting studies done on a much larger scale also reveal a similar trend with comparable numbers of both unique and shared fingerprints as shown in the table.

To compare this user data with crawler fingerprints, we first updated our crawler fingerprints. For this, we used *PhishPrint* to re-generate 50 new token URLs per crawler and solicited scans again which yielded 57 crawler fingerprints. By combining this with data from two prior experiments (Table 1, §3.2.1), we obtained a total of 256 distinct crawler fingerprints from across a period of 13 months. Comparing this with MTurk data, we found that 137 users (13.6%) had one of 32 fingerprints that were colliding with the crawlers. Of them, more than 90% of the users had a shared fingerprint thus indicating that most of the collisions were due to those fingerprints which are already common among the users themselves. The breakdown of this data by OS is reported in Table 7.

When analyzing the fingerprint specificity results, it is important to note how all the fingerprint numbers continue to increase as the scale of the study increases. For example, in [22], about 2 million users had 78,037 distinct and 12,250 shared fingerprints. On the other hand, the number of fingerprints collected from all 23 crawlers (including the VT ecosystem

	Canvas (s)	Font (s)	Total (s)
Mean	0.09	3.97	4.26
Median	0.06	2.36	2.52
90%	0.16	8.88	9.4

Table 6: Time taken for obtaining BFPs during user study

containing 80 crawlers) across a period of one year is only 256. Thus, in the worst case, even if all 256 of these fingerprints end up as shared fingerprints in a larger-sized user study, we can expect a significant population of victims to remain vulnerable to the proposed attacks. However, this trend of increase in fingerprints as the study scale increases points to the need for a much larger user study to accurately assess the specificity of fingerprints. While such a dedicated large-sized user study is outside our means, in §3.2.1, we showed how we used AmIUnique's data of 467K users for directly computing the fingerprint collisions between crawlers and potential victims.

Further, we found that Bitdefender, PhishTank, SmartScreen, APWG and GSB are the main five crawlers associated with the collisions accounting for 45%, 43%, 28%, 7% and 6% of the collisions respectively. Altogether, these five crawlers account for 98% of the collisions (134 users). Note that these are the same five crawlers that we have already studied in our AmIUnique-based specificity experiment on a much larger scale (§3.2.1).

Timing Analysis. One might also argue that such sophisticated fingerprinting based cloaking logic will result in a computational time delay that can reduce the effectiveness of social engineering attacks launched on real users. In order to see if this is true, we measured the time required to perform the cloaking logic for the users in our first user study. Our results (Table 6) show that most of the time is spent in obtaining font fingerprints with mean time for the cloaking logic being 4.26 seconds. At first, it appears that it might be possible to reduce the cloaking logic time by using a "progressive" logic such as extracting and checking the faster fingerprints first before going to the slower ones. But this is not productive for cloaking attacks as victim machines will progress all the way to the end of the cloaking logic anyway. However, given that the mean time to fully load a web page on a desktop machine is about 10.4 seconds [8], attackers can use that to their advantage. As our fingerprinting code is very light in size (Appendix D), an attacker can potentially load and start to run it immediately while simultaneously "pretending" to be loading a site. This way, the attacker can gain a sufficient compute time budget for the cloaking logic.

5 Countermeasures

The *CVD Scores* reported in Table 1 can serve as a "report card" for crawlers trying to prioritize their mitigation efforts across different areas of weaknesses as we discuss below.

Browser Anomalies. At the outset, this seems like a simple question of applying best practices as some crawlers already have near-perfect scores. However, this is only true to a certain extent. One issue is that many crawlers process a large number of URLs daily. Hence, it is common practice to use headless browsers for scalability [16]. However, this results in an arms race[9] between such browsers and their detectors [25]. While our rudimentary crawler artifact vectors did not take such sophisticated headless browser detection features into account, it should be trivial to include them in *PhishPrint* profiling pages and come up with a much more sophisticated anomaly cloaking vector. Further, biometric behavior-based bot detection systems can further complicate this issue for crawlers [18] opening room for new evasion vectors. While handling all these issues might involve elaborate browser changes, ML-driven crawler behavior, and/or scalability compromises, we suggest the vendors to prioritize on fixing the simpler issues. All crawlers should visit each URL atleast once with a "Real Browser" that supports all web APIs and try to hide well-known artifacts [26, 45, 46].

Network Data. For handling these weaknesses, crawlers have to diversify their network infrastructure in terms of both IP addresses as well as geographical diversity and using residential networks. GSB and PhishTank are some of the best examples of this. However, during our vulnerability disclosure, some companies have mentioned that it might be difficult for them to address this due to financial implications. In these cases, we suggest that vendors consider approaches such as using peer-to-peer VPN networks [7] and sharing URLs with other crawlers to help improve network diversity.

Advanced Fingerprints. Our study found that there was extremely limited diversity of `<F,C,W>`s across the entire ecosystem. The maximum `<F,C,W>` blocklist CVD score across all crawlers was only 9.3 with several crawlers having less than 10 distinct `<F,C,W>`s across hundreds of scanned URLs. Among the three individual fingerprinting vectors of `<F,C,W>`, improving font diversity is the easiest to fix as it only needs increasing the number of "font sets" installed in the crawler instances. When doing this, it should be ensured that the fonts match the general font set characteristics of users from that geolocation. Some vendors already started doing this as a result of our disclosures. However, the Canvas and WebGL fingerprints require more intricate mitigations. Currently, there are 3 approaches for this:

- **Blocking.** [24] proposed an ML-based script blocking solution for fingerprinting code. However, such solutions cannot be used by crawlers as the presence of such blocking can itself be used for evasion. Instead of blocking, URLs can be isolated for further automated/human analysis. However, the attackers can even fingerprint such analysts' browsers and add their fingerprints to their blocklists. While victim-side blocking solutions might still work, the problem here is that of deployment. Unless such a client-side solution is baked into all major browsers, it might not achieve good coverage. This problem is further exacerbated

[9]At the time of writing this manuscript in October 2020, unfortunately the headless detectors seem to be winning this battle.

by the fact that many phishing victims may also be slow adopters for technologies such as security extensions.

- **Uniformity.** Uniform software re-rendering approaches that result in the same fingerprints for all users have also been proposed [50]. However, these also have the same coverage issues as above. Unless, a majority of all users adopt the same solution, the resulting uniform fingerprint can itself be used as an evasion vector while not losing victims.
- **Randomization.** This works by randomizing the fingerprints in each browsing session [35]. Brave browser has adopted this to devise a solution for Canvas and WebGL fingerprinting by adding small random noise to the generated data [31]. This is the most promising approach for crawlers as it does not need to be adopted universally for this to work.

Hence, we recommend vendors to adopt similar *transparent* randomization-based defenses in order to defend against Canvas and WebGL-based fingerprinting attacks. Another possible solution is to use dynamic software reconfiguration approaches [28], although these have scalability limitations.

URL Reporting. It is also important for all the crawler vendors to prevent abuse of their reporting infrastructure. In this research, by simply registering a single domain and self-reporting its wild card subdomains, we were able to collect a large amount of sensitive information such as fingerprints and IP addresses of many crawlers. We discussed early on in the paper about how segregating profilable infrastructure based on candidate domains will work in an attacker's advantage (§1: TLD+1 Bias). However, the crawlers can at least use such separation techniques to divide their limited crawler resources between submissions from vetted and non-vetted URL reporters. Crawler vendors can also leverage existing spam and anomaly detection research work to monitor and detect abuse of URL reporting services and prevent anomalous submissions of token URLs for profiling of crawlers.

Some vendor-specific recommendations we make are in Appendix B.

6 Discussion

Vulnerability Disclosure. We completed an effective vulnerability disclosure process. We submitted detailed vulnerability reports to all 23 security crawlers (21 vendors) that we have specifically profiled. 9 vendors (10 crawlers) have so far acknowledged our results including Google (GSB), Microsoft (SmartScreen, Outlook), Norton, AlienVault and Sophos. We had follow-up discussions over e-mail and online meetings with 7 vendors on our results. Of the 9 vendors, 3 mentioned that they were already working on changes or were aware of these limitations. 6 of them have reported to be working on follow-up changes with one vendor mentioning about having tasked multiple engineers to work on the problems we pointed out in our paper. We also received a Google Vulnerability Reward for our research. Our reward amount was the highest in the category of "abuse-related methodologies" indicating both "High Impact" and "High Probability" [9] of the cloaking attacks we discovered. As a follow up, we also received three Vulnerability Research Grants from Google encouraging us to continue studying their security crawlers in the future.

Single TLD+1 Bias. One might argue that using multiple subdomains under a single TLD+1 will deliver a lot less diverse profiling information from crawlers than using multiple TLD+1s. However, we show in §4.1.2 that the diverse profiling information we collect from a single `.com` domain generalizes well enough to "protect" phishing pages hosted on 20 `.xyz` domains. We also performed a small control experiment using 5 `.xyz` domains which confirms the same (§3.2.2). The extensive positive feedback we received from the crawler vendors during vulnerability disclosure also attests to the sensitivity of the information we were able to gather by using a single registered domain. Most importantly, we argue that if a crawler were to choose to segregate their "profilable" infrastructure based on domain names, then it would only end up making an attacker's job easier. This is because the attacker can then begin to first use a candidate domain name in a benign mode for quickly collecting the limited profile of the segregated crawling infrastructure. They can then switch that same domain into a malicious mode by hosting phishing content hidden with the help of forensic information found during the profiling stage.

Limitations. While *PhishPrint* evaluates crawlers by avoiding phishing experiments, there are some use cases where these experiments are indispensable. For example, prior works such as [39] that focused on speed of population of browser blocklists and [40] that focused on dynamic label changes of URLs can only be accomplished with the help of phishing experiments. Furthermore, our system will be unable to measure crawler resilience against dynamic cloaking attacks such as the "Timing" attacks studied in [52] as we are limited to only the profiling data that can be captured from the crawlers. Nevertheless, *PhishPrint* presents a scalable solution to measure a wide variety of weaknesses of crawlers and thus can be considered complementary to existing phishing experiment-based designs.

We also note that the measurements such as "# IPs" that we presented in Table 1 could be overcounted due to the presence of URL sharing between crawlers (except when such counts are 1 or 0). This is a difficult problem to solve given that there might be a lot of undisclosed URL sharing happening between various security entities. However, it is important to recognize that this only means that our measurements might over-estimate a crawler's infrastructure. This means that the weaknesses of crawlers could in reality be more than measured. We experienced this during disclosure when certain entities have conceded that the actual number of IP addresses that they own is less than what we showed in our report.

Finally, we would like to point out the "double-edged sword" nature of *PhishPrint*. While it can allow researchers to study crawlers in a low-cost, highly scalable manner, at the same time, it might allow attackers to host long-lasting evasive malicious websites at a low-cost. For this purpose, we have made recommendations to monitor abuse of reporting APIs to all crawlers in §5. If such monitoring does come into effect as a result of this study, we would welcome that as another positive security outcome. Furthermore, security researchers can still seek special permissions to bypass such monitors and continue their evaluation of crawlers in a low-cost manner.

Future Work. Given the low-cost and scalable nature of *PhishPrint*, we would like to continue to use it to study more cloaking vectors. In the future, we would like to study the resilience of crawlers against some other fingerprinting vectors such as `MediaDevices`, `Web Audio` and `Battery` and `Sensor` Web APIs. Furthermore, we also want to study the potential of developing ML-driven cloaking attacks using the behavioral biometrics aspects of crawlers.

Ethical Considerations. Our 70-day profiling study resulted in submitting about 840 token URLs to each crawler at the rate of 12 URLs per day. During the 2-week period when our 20 phishing URLs were reported as well, this number went up to 32 per crawler per day. While we concede that the time spent in scanning these URLs is a waste for the crawlers, we argue that this number is very small in comparison to the huge number of URLs they receive each day. We have also disclosed our URL submission frequency to all crawlers. Moreover, our method of submitting token URLs to crawlers to gain insights is similar to some prior works [37,39,40]. We assess the impact of our token site submissions with PhishTank as an example. With the help of PhishTank's web portal we were able to determine that our token URLs from both experiments accounted for less than 0.8% of their total received URLs during that period. We argue that the security benefits gained by the measurements from our study far outweigh this minor overhead that the crawler vendors experienced during our experimentation period. Some vendors have explicitly mentioned the same and asked us to continue the study and share new insights in the future. With regards to the simulated phishing websites used during the experiments, we did not share those URLs with any human users and only submitted them to the crawlers. We also made sure that they are completely non-functional by removing all form submit buttons in order to prevent effects of accidental exposure to users. Similar efforts were also made previously [37,40]. Finally, we also obtained IRB exemptions for both our user studies.

7 Related Work

The closest works to *PhishPrint* are [32, 37] and partially [39, 52] as all of them involved evaluating security crawlers against cloaking attacks using simulated phishing sites. In our research, we proposed an alternate highly scalable solution that avoids phishing sites and instead directly relies on profiling the crawlers to find new cloaking weaknesses. In Appendix C, we show that this alternate approach can capture the same measurements as prior works. However, as discussed in §6 (Limitations), while phishing experiments work in all contexts, *PhishPrint* is restricted to measuring only those cloaking weaknesses that can be gleaned from the passive profiling information extracted from crawlers. As a result, our design can be considered a complement that can co-exist with the current phishing site-based approaches.

In terms of cloaking weaknesses, recent works such as [32] and partially [52] have focused on testing the resilience of crawlers against cloaking attacks powered by CAPTCHAs, human-interaction detectors and basic browser fingerprinting techniques such as Cookies and Referer headers. In our work, we found wide-spread anomalies in crawlers such as artifacts that give away signs of browser automation and incapacity to execute advanced Web API code. We also found great limitations in the diversity of network infrastructure (IP, AS space) as well as advanced fingerprints (Canvas, WebGL and JS-based Font) associated with crawlers. We developed new cloaking attacks from these weaknesses. Note that *PhishPrint* can also be easily re-deployed to evaluate crawlers against many of these cloaking attacks in [32,52] (except timing attacks) in the future. Many research works have focused on studying in-the-wild cloaking and evasive techniques [23,38,39,42,49,52] which was not our focus.

In order to collect and analyze the profiling data from crawlers, we applied techniques studied in prior works. [45,46] have described techniques to detect browser automation indicators and anomalies of privacy-preserving browsers which we applied in our study to discover artifacts of crawlers. Further, we also successfully applied advanced browser fingerprinting techniques described and developed in [29, 36] to capture crawler fingerprints. We also relied on other works in browser fingerprinting to analyze the specificity [22, 29, 47] and propose suitable countermeasures [19,28,30,35,43,44] for the crawlers. On a related note, while we measured the applicability of fingerprinting to launch attacks on security crawlers, some recent works have focused on a complementary question of how fingerprinting can be used to yield security benefits [13,27,44,48].

8 Conclusion

We built a novel, scalable, low-cost framework named *PhishPrint* to enable the evaluation of web security crawlers against multiple cloaking attacks. *PhishPrint* is unique in that it completely avoids the use of any simulated phishing sites and instead relies on benign profiling pages. We used *PhishPrint* to evaluate 23 crawlers in a 70-day study which found several previously unknown cloaking weaknesses across the crawler ecosystem. We confirmed the practical impact of our findings by deploying evasive phishing web pages and performing user studies. We also discussed concrete mitigation measures in areas of crawling and reporting infrastructures. We have relayed the found weaknesses to the crawler vendors through a vulnerability disclosure process that resulted in some remedial actions as well as multiple vulnerability rewards.

Acknowledgements

We thank Anish Chand for working on an early prototype of *PhishPrint* as a proof of concept. We also like to acknowledge Julian Gale and Christopher Martin for helping in building the Web Scan Requestor module. We convey our thanks to Roberto Perdisci for providing us with valuable research advice throughout the project. We are grateful to all the concerned PC members at IEEE SSP 2021 and Usenix Security 2021 for providing detailed constructive feedback. Finally, we thank Pierre Laperdrix and the AmIUnique project team for enabling access to crucial browser fingerprinting data and code. This work was done with help of funds from UNO Office of Research and UNO Tolmas scholars program whose support we hereby gratefully acknowledge.

References

[1] Advanced outlook.com security for office 365 subscribers. https://web.archive.org/web/20200901032551/https://support.microsoft.com/en-us/office/advanced-outlook-com-security-for-office-365-subscribers-882d2243-eab9-4545-a58a-b36fee4a46e2.

[2] Amiunique. https://amiunique.org.

[3] Browser market share worldwide. https://gs.statcounter.com/browser-market-share.

[4] Browserling. https://www.browserling.com/.

[5] Google safe browsing block all my subdomains instead only effected one. https://support.google.com/webmasters/thread/17514260?hl=en.

[6] Google safe browsing erroneously blocking my whole domain and subdomains. https://support.google.com/webmasters/thread/32022154?hl=en.

[7] Hola better internet – access censored sites. https://hola.org/faq.

[8] Page load times. https://backlinko.com/page-speed-stats.

[9] Program rules – application security: "reward amounts for abuse-related methodologies". https://www.google.com/about/appsecurity/reward-program/.

[10] Teams powers office 365 growth. https://office365itpros.com/2020/04/30/office365-teams-power-growth/.

[11] Virustotal. https://www.virustotal.com/gui/.

[12] Xyz domain name policies. https://nic.monster/files/XYZ-registry-domain-name-policies.pdf?v=2.0.

[13] Furkan Alaca and Paul C. van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *ACSAC 2016*.

[14] APWG. Phishing activity trends report: 3rd quarter, 2019. https://docs.apwg.org/reports/apwg_trends_report_q3_2019.pdf.

[15] Michael Archambault. Microsoft security reports a massive increase in malicious phishing scams. https://www.digitaltrends.com/computing/microsoft-security-massive-increase-phishing-scams/.

[16] Eric Bidelman. Getting started with headless chrome. https://developers.google.com/web/updates/2017/04/headless-chrome, Jan 2019.

[17] Yinzhi Cao, Song Li, and Erik Wijmans. (cross-)browser fingerprinting via OS and hardware level features. In *NDSS 2017*.

[18] Zi Chu, Steven Gianvecchio, Aaron Koehl, Haining Wang, and Sushil Jajodia. Blog or block: Detecting blog bots through behavioral biometrics. *Comput. Networks*, 2013.

[19] Amit Datta, Jianan Lu, and Michael Carl Tschantz. Evaluating anti-fingerprinting privacy enhancing technologies. In *WWW 2019*, pages 351–362.

[20] MDN Web Docs. Canvas api. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.

[21] MDN Web Docs. Webgl: 2d and 3d graphics for the web. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API.

[22] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. Hiding in the crowd: an analysis of the effectiveness of browser fingerprinting at large scale. In *WWW 2018*, pages 309–318.

[23] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean Michel Picod, and Elie Bursztein. Cloak of visibility: Detecting when machines browse a different web. In *IEEE Symposium on Security and Privacy, SP 2016*, pages 743–758.

[24] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. *CoRR*, abs/2008.04480, 2020.

[25] Paul Irish. paulirish/headless-cat-n-mouse. https://github.com/paulirish/headless-cat-n-mouse, Jan 2018.

[26] Jordan Jueckstock and Alexandros Kapravelos. Visiblev8: In-browser monitoring of javascript in the wild. In *IMC 2019*, pages 393–405.

[27] Pierre Laperdrix, Gildas Avoine, Benoit Baudry, and Nick Nikiforakis. Morellian analysis for browsers: Making web authentication stronger with canvas fingerprinting. In *DIMVA 2019*, pages 43–66.

[28] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. Fprandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *ESSoS 2017*, pages 97–114.

[29] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *IEEE Symposium on Security and Privacy, SP 2016*, pages 878–894.

[30] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Mitigating browser fingerprint tracking: Multi-level reconfiguration and diversification. In *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*, pages 98–108.

[31] Peter Snyder Mark Pilgrim and Ben Livshits. Fingerprint randomization. https://web.archive.org/web/20200728132011/https://brave.com/whats-brave-done-for-my-privacy-lately-episode3/.

[32] Sourena Maroofi, Maciej Korczynski, and Andrzej Duda. Are you human?: Resilience of phishing detection to evasion techniques based on human verification. In *IMC 2020*, pages 78–86.

[33] Angela Moscaritolo. Beware: Phishing attacks are on the rise. https://www.pcmag.com/news/beware-phishing-attacks-are-on-the-rise.

[34] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP*, pages 1–12, 2012.

[35] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *WWW 2015*, pages 820–830.

[36] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *2013 IEEE Symposium on Security and Privacy, SP 2013*, pages 541–555.

[37] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In *IEEE Symposium on Security and Privacy, SP 2019*, pages 1344–1361.

[38] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Gary Warner. Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In *eCrime 2018*, pages 1–12.

[39] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. Phishtime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *USENIX Security 2020*, pages 379–396.

[40] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *IMC 2019*, pages 478–485.

[41] Peter Snyder and Ben Livshits. Brave, fingerprinting, and privacy budgets. https://web.archive.org/web/20200809060950/https://brave.com/brave-fingerprinting-and-privacy-budgets/.

[42] Ke Tian, Steve T. K. Jan, Hang Hu, Danfeng Yao, and Gang Wang. Needle in a haystack: Tracking down elite phishing domains in the wild. In *IMC 2018*, pages 429–442.

[43] Christof Ferreira Torres, Hugo L. Jonker, and Sjouke Mauw. Fp-block: Usable web privacy by controlling browser fingerprinting. In *ESORICS 2015*, pages 3–19.

[44] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupé. Everyone is different: Client-side diversification for defending against extension fingerprinting. In *USENIX Security 2019*, pages 1679–1696.

[45] Phani Vadrevu and Roberto Perdisci. What you see is NOT what you get: Discovering and tracking social engineering attack campaigns. In *IMC 2019*, pages 308–321.

[46] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. Fp-scanner: The privacy implications of browser fingerprint inconsistencies. In *USENIX Security 2018*, pages 135–150.

[47] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. FP-STALKER: tracking browser fingerprint evolutions. In *IEEE Symposium on Security and Privacy, SP 2018*, pages 728–741.

[48] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. Fp-crawlers: Studying the resilience of browser fingerprinting to block crawlers. In *MADWeb 2020*.

[49] David Y. Wang, Stefan Savage, and Geoffrey M. Voelker. Cloak and dagger: dynamics of web search cloaking. In *CCS 2011*, pages 477–490.

[50] Shujiang Wu, Song Li, Yinzhi Cao, and Ningfei Wang. Rendered private: Making GLSL execution uniform to prevent webgl-based browser fingerprinting. In *USENIX Security 2019*, pages 1645–1660.

[51] Katsunari Yoshioka, Yoshihiko Hosobuchi, Tatsunori Orii, and Tsutomu Matsumoto. Your sandbox is blinded: Impact of decoy injection to public malware analysis systems. *J. Inf. Process.*, 19:153–168, 2011.

[52] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kapravelos, Tiffany Bao, Ruoyu Wang, et al. Crawlphish: Large-scale analysis of client-side cloaking techniques in phishing. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2021.

A Breakdown of MTurk Study Results

Table 7 breaks down the results of our second user study described in §4.2 by OS. It is to be noted that the sum of values in the third, fourth and fifth columns do not add up to the values in the final row. This is because of a small amount of overlap in the fingerprints across different platforms. The final column shows the breakdown of the 137 collisions that were seen with the crawlers' fingerprints.

OS	# Users	# Distinct	# Unique	# Shared	Norm. Entropy	# Collide
Windows	693	425	344	81	0.866	115
Chrome OS	35	17	12	5	0.707	8
Linux	16	16	16	0	1.0	0
iOS	29	10	6	4	0.568	0
Mac OS X	146	87	68	19	0.824	8
Android	88	41	26	15	0.731	6
All	1007	592	469	123	0.865	137

Table 7: OS-based breakdown of fingerprints collected from our second user study

B Specific Recommendations

During our profiling study, we saw some specific problems with PhishTank and GSB that are discussed below along with suitable recommendations.

B.1 PhishTank

PhishTank shows the reported URLs on their website to allow human analysts to investigate them. We found a couple of serious issues with PhishTank's web portal ecosystem that are described below:

1. We noticed that repeated URL submissions are ignored by PhishTank and not shown in their homepage even if the URL is being re-submitted from a different user account. An attacker can exploit this by simply self-reporting their URLs to PhishTank a few days before adding malicious content to them. This will effectively prevent the URL from ever showing up on the homepage and thus reduce the potential variety of visitors to which the website will get exposed. To prevent this, PhishTank should bump up URLs to their homepage whenever they get resubmitted by a different user account.

2. We noticed that PhishTank allows their website visitors to open and check the new URLs either in a new window or in an `iframe` in PhishTank. However, in both cases, it is possible for an attacker to check if the `Referer` points to `phishtank.com` and trigger benign behavior. We have used this same evasion logic in our experiments. Thus, unless a human analysts copies the URL and pastes it in their URL address bar, it will always carry the `Referer` artifact, thus making it easy for an attacker to decide to cloak and evade manual analysis. Hence, we strongly recommend PhishTank to use `Referrer-Policy` headers (for example, by setting it to `same-origin`) to combat such evasion strategies.

B.2 Google Safe Browsing

During the initial setup phase of our longitudinal study, we saw a couple of serious issues with Google Safe Browsing's (GSB) crawler infrastructure. As these are specific to GSB, we are reporting them separately here.

1. We noticed that GSB's infrastructure was restricting large-sized data packets from being shipped out of their network hosting their crawlers. For example, we were unable to ship a 50 KB sized packet from our client code running in the crawlers' browsers to our servers. This was a peculiar restriction that we did not notice with any other crawler vendor. As an attacker can easily abuse such properties for evasion, we recommend GSB to re-consider such restrictions.

2. Further, we noticed that while all other crawlers take at least a couple of seconds to execute our fingerprinting scripts, GSB's crawlers were able to do this in less than 30 milliseconds. Our preliminary manual testing with many popular web browsers also showed that it takes at least two seconds to execute this code. Attackers can thus use such timing discrepancies to detect the presence of a powerful JavaScript execution framework and trigger their cloaking logic. We did not need to include these timing-based side channels in our cloaking logic as we were already able to handle GSB and other crawlers by capitalizing on their limited fingerprint diversity.

C Evolution of Security Crawlers

As mentioned previously, *PhishPrint* is a crawler evaluation framework with an alternate non-phishing based design that can conduct the evaluation of security crawlers against many cloaking attacks that were done by prior works. In order to demonstrate this, we use the profiling data we obtained from crawlers during our 70-day study. Using this data, we attempt to repeat the measurements made by authors of *PhishFarm* [37]. This way we can study how the crawlers have evolved from the time of their study to ours.

PhishFarm studied the effectiveness of four user agent-based cloaking vectors (called as Filters B, C, D and F) and 1 blocklist-based cloaking vector (called Filter E) against five crawlers. Four of those crawlers overlap with our work: APWG, GSB, SmartScreen and PhishTank. So, we consider these four crawlers here. Filter B serves malicious traffic to only mobile user agents. Filters C and D serve malicious traffic to US and non-US based clients that use Desktop GSB browsers (Chrome, Firefox or Safari). Filter F is equivalent to the JS execution anomaly vector (as it is tied to a JS `onload` event execution). We were unable to report about Filter E as it uses a specific `.htaccess` file for blocklisting for which we do not have any access. Also, Filter A is a control filter and can hence be ignored here. By analyzing the HTTP headers and IP addresses in our collected profiling data, we were able to gauge how well the crawlers would have defended against filters: B, C, D and F if they were deployed in our reported URLs.

Table 8 shows the results. The CVD scores for the four vectors are shown in the four columns. The scores are shown as fractions here in order to enable direct comparison with results from [37] which reported the scores on a scale of 0 to 1. In the *PhishFarm* study, it was reported that except for Filter B, all the other filters would be defended against by one

Crawler	Mobile (B)	Desktop - GSB US (C)	Desktop GSB Non-US (D)	Real Browser (F)
APWG	1.000	0.942	0.690	1.000
GSB	0.000	0.347	0.741	0.914
SmartScreen	0.000	0.001	0.075	0.989
PhishTank	0.992	0.998	0.998	1.000

Table 8: CVD scores for the cloaking vectors studied in [37]

of the crawlers. Further, it was mentioned that after the study, improvements have been made for defending against Filter B as well. Our study confirms these results. Compared to their study, both APWG and PhishTank have massively improved with respect to Filters B, C and D thus indicating that they have begun to use mobile user agents as well as GSB-based desktop user-agents worldwide. However, unfortunately, SmartScreen and GSB still do not adequately scan from mobile user agents. Further, SmartScreen continues to perform badly on both filters C and D. We investigated this and found that this is because they mostly used IE-based web browser agents which the filter explicitly avoids.

D Browser Fingerprinting Code

We provide below the JavaScript code snippets for Canvas, WebGL and Font Fingerprinting that we adapted from *AmIUnique* for profiling the crawlers.

```

function generate_canvas_data() {
    try {
        var canvas = document.createElement('canvas');
        canvas.height = 60;
        canvas.width = 400;
        var canvasContext = canvas.getContext('2d');
        canvas.style.display = 'inline';
        canvasContext.textBaseline = 'alphabetic';
        canvasContext.fillStyle = '#f60';
        canvasContext.fillRect(125, 1, 62, 20);
        canvasContext.fillStyle = '#069';
        canvasContext.font = '11pt no-real-font-123';
        canvasContext.fillText
               ("Cwm fjordbank glyphs vext quiz, \uD83D\uDE03", 2, 15);
        canvasContext.fillStyle = 'rgba(102, 204, 0, 0.7)';
        canvasContext.font = '18pt Arial';
        canvasContext.fillText
               ("Cwm fjordbank glyphs vext quiz, \uD83D\uDE03", 4, 45);
        canvasData = canvas.toDataURL();
        return canvasData;
    } catch (e) {
        canvasData = 'Not supported';
        return canvasData;
    }
}
```

Listing 1: Canvas Fingerprinting Code

```

function generate_web_gl_data() {
    try {
        var gl = canvas.getContext
               ('webgl') || canvas.getContext('experimental-webgl');
        var vShaderTemplate = 'attribute vec2
                attrVertex;varying vec2 varyinTexCoordinate;uniform vec2
                uniformOffset;void main(){varyinTexCoordinate=attrVertex
               +uniformOffset;gl_Position=vec4(attrVertex,0,1);}';
        var fShaderTemplate = 'precision
                mediump float;varying vec2 varyinTexCoordinate;void
                main() {gl_FragColor=vec4(varyinTexCoordinate,0,1);}';
        var vertexPosBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, vertexPosBuffer);
        var vertices = new Float32Array
               ([-.2, -.9, 0, .4, -.26, 0, 0, .732134444, 0]);
        gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
        vertexPosBuffer.itemSize = 3;
        vertexPosBuffer.numItems = 3;
        var program = gl.createProgram();
        var vshader = gl.createShader(gl.VERTEX_SHADER);
        gl.shaderSource(vshader, vShaderTemplate);
        gl.compileShader(vshader);
        var fshader = gl.createShader(gl.FRAGMENT_SHADER);
        gl.shaderSource(fshader, fShaderTemplate);
        gl.compileShader(fshader);
        gl.attachShader(program, vshader);
        gl.attachShader(program, fshader);
        gl.linkProgram(program);
        gl.useProgram(program);
        program.vertexPosAttrib
               = gl.getAttribLocation(program, 'attrVertex');
        program.offsetUniform
               = gl.getUniformLocation(program, 'uniformOffset');
        gl.enableVertexAttribArray(program.vertexPosArray);
        gl.vertexAttribPointer(program.vertexPosAttrib
           , vertexPosBuffer.itemSize, gl.FLOAT, !1, 0, 0);
        gl.uniform2f(program.offsetUniform, 1, 1);
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, vertexPosBuffer.numItems);

        if (gl.canvas != null) {
            return gl.canvas.toDataURL();
        }
        else {
             return 'Not supported';
        }

    } catch (e) {
        return 'Not supported';
    }
}
```

Listing 2: WebGL Fingerprinting Code

```

function get_font_list() {
    var baseFonts = ['serif', 'sans-serif', 'monospace'];
    // Below is a test font list containing 1043 fonts.
    var testFonts = ['.Aqua Kana', '.Helvetica LT MM', .... 'ori1Uni'];
    var testSize = '72px';
    var testChar = 'A';
    var h = document.getElementById('font');

    // Get the width of the text by creating a span
    var s = document.createElement('span');
    s.style.fontSize = testSize;
    s.innerText = testChar;
    var defaultFonts = {};

    for (var indexBaseFonts in baseFonts) {
        baseFont = baseFonts[indexBaseFonts];
        s.style.fontFamily = baseFont;

        if (h) {
            h.appendChild(s);
            defaultFonts[baseFont] = {};
            defaultFonts[baseFont]['offsetWidth'] = s.offsetWidth;
            defaultFonts[baseFont]['offsetHeight'] = s.offsetHeight;
            h.removeChild(s);
        }
    }

    fontsDetected = {};

    for (var indexFont in testFonts) {
        font = fonts[indexFont];
        detected = false;
        fontStyle = '"' + font + '"';

        for (var indexBaseFonts in baseFonts) {
            baseFont = baseFonts[indexBaseFonts];
            // Append base font at the end of test font for fallback
            s.style.fontFamily = fontStyle + ',' + baseFont;

            if (h) {
                h.appendChild(s);
                var match = s.offsetWidth != defaultFonts
                       [baseFont]['offsetWidth'] || s.offsetHeight
                        != defaultFonts[baseFont]['offsetHeight'];
                h.removeChild(s);
                detected = detected || match;

                if (detected) {
                    break;
                }
            }
        }

        fontsDetected[font] = detected;
    }

    return fontsDetected;
}
```

Listing 3: Font List Fingerprinting Code