Cloak of Visibility: Detecting When Machines Browse A Different Web

Luca Invernizzi*, Kurt Thomas*, Alexandros Kapravelos†,
Oxana Comanescu*, Jean-Michel Picod*, and Elie Bursztein*
*Google, Inc. {invernizzi, kurtthomas, elieb, oxana, jmichel}@google.com
†North Carolina State University kapravelos@ncsu.edu

***Abstract*—The contentious battle between web services and miscreants involved in blackhat search engine optimization and malicious advertisements has driven the underground to develop increasingly sophisticated techniques that hide the true nature of malicious sites. These *web cloaking* techniques hinder the effectiveness of security crawlers and potentially expose Internet users to harmful content. In this work, we study the spectrum of blackhat cloaking techniques that target browser, network, or contextual cues to detect organic visitors. As a starting point, we investigate the capabilities of ten prominent cloaking services marketed within the underground. This includes a first look at multiple IP blacklists that contain over 50 million addresses tied to the top five search engines and tens of anti-virus and security crawlers. We use our findings to develop an anti-cloaking system that detects split-view content returned to two or more distinct browsing profiles with an accuracy of 95.5% and a false positive rate of 0.9% when tested on a labeled dataset of 94,946 URLs. We apply our system to an unlabeled set of 135,577 search and advertisement URLs keyed on high-risk terms (e.g., luxury products, weight loss supplements) to characterize the prevalence of threats in the wild and expose variations in cloaking techniques across traffic sources. Our study provides the first broad perspective of cloaking as it affects Google Search and Google Ads and underscores the minimum capabilities necessary of security crawlers to bypass the state of the art in mobile, rDNS, and IP cloaking.**

I. Introduction

The arms race nature of abuse has spawned a contentious battle in the realm of *web cloaking*. Here, miscreants seeking to short-circuit the challenge of acquiring user traffic turn to search engines and advertisement networks as a vehicle for delivering scams, unwanted software, and malware to browsing clients. Although crawlers attempt to vet content and expunge harmful URLs, there is a fundamental limitation to browsing the web: not every client observes the same content. While *split views* occur naturally due to personalization, geo optimization, and responsive web design, miscreants employ similar targeting techniques in a blackhat capacity to serve enticing and policy-abiding content exclusively to crawlers while simultaneously exposing victims to attacks.

Where as a wealth of prior work focused on understanding the prevalence of cloaking and the content behind cloaked doorways, none precisely measured the spectrum of cloaking techniques in the wild as it affects search engines and ad networks. Indeed, earlier studies predicated their analysis on a limited set of known cloaking techniques. These include redirect cloaking in search engine results [16], [18], [24], [27], [33], [34] or search visitor profiling based on the `User-Agent` and `Referer` of HTTP requests [32], [35]. An open question remains as to what companies and crawlers blackhat cloaking software targets, the capabilities necessary for security practitioners to bypass state of the art cloaking, and ultimately whether blackhat techniques generalize across traffic sources including search results and advertisements.

In this paper, we marry both an underground and empirical perspective of blackhat cloaking to study how miscreants scrutinize an incoming client's browser, network, and contextual setting and the impact it has on polluting search results and advertisements. We root our study in the blackmarket, directly engaging with specialists selling cloaking software. In total, we obtain ten cloaking packages that range in price from $167 to $13,188. Our investigation reveals that cloaking software spans simple Wordpress plugins written in PHP that check the User-Agent of incoming clients, to fully customized forks of the Nginx web server with built-in capabilities for blacklisting clients based on IP addresses, reverse DNS, User-Agents, HTTP headers, and the order of actions a client takes upon visiting a cloaked webpage. We also obtain access to multiple IP blacklist databases, one of which covers 54,166 IP addresses associated with Bing, Yahoo, Google, Baidu, and Yandex, and a second that contains over 50 million IP addresses from universities (e.g., MIT, Rutgers), security products (e.g., Kaspersky, McAfee), VPNs, and cloud providers. Our analysis yields a unique perspective of which web services miscreants seek to evade and the technical mechanisms involved.

We leverage our tear-down of blackhat cloaking techniques to build a scalable de-cloaking crawler and classifier that detects when a web server returns divergent content to two or more distinct browsing clients. We fetch content from 11 increasingly sophisticated user emulators that cover a combination of Chrome, Android, and a simple robot accessing the Internet via residential, mobile, and data center networks including one associated with Google's crawling infrastructure. In total, we crawl 94,946 labeled training URLs multiple times from each profile, totaling over 3.5 million fetches. We then build a classifier that detects deviations in the content, structure, rendering, linguistic topics, and redirect graph between all pairs of crawlers, accurately distinguishing blackhat cloaking from mobile and geo targeting with 95.5% accuracy and a false positive rate of 0.9%. We analyze in depth which features and browser profiles are critical to detecting cloaking, finding no single approach covers all cloaking techniques.

We apply our system to an unlabeled set of 135,577 Google Search and Google Ads URLs keyed on high-risk terms commonly targeted by miscreants (e.g., luxury products, weight loss supplements) and find 11.7% of the top 100 search results and 4.9% of ads cloak against the Googlebot crawler. Of these, the most popular blackhat cloaking techniques involve detecting JavaScript, blacklisting Googlebot's User-Agent and IP, and requiring that visitors interact with content before the server finally delivers the de-cloaked payload. Despite a menagerie of cloaking techniques in the wild that vary drastically between search and ads, our system nevertheless succeeds at generalizable detection. We dig into specific case studies and their monetization approaches, revealing a thriving market that attempts to capitalize on legitimate consumer interest in nutraceuticals, mobile gaming, and online shopping.

Finally, we explore the fragility of de-cloaking systems, including our own, to miscreant's adapting their cloaking techniques. Rather than persist in the arms race to defeat increasingly sophisticated browser fingerprinting techniques, we argue our approach of comparing the content that cloaked servers deliver to multiple browsing clients naturally extends to real rather than emulated clients. We discuss the potential for client-side detection of cloaking as well as centralized reporting and scoring. Both of these approaches hinder the ability of malicious servers to show benign content exclusively to crawlers, though their adoption must overcome potential privacy concerns.

In summary, we frame our contributions as follows:

- We provide the first broad study of blackhat cloaking techniques and the companies affected.
- We build a distributed crawler and classifier that detects and bypasses mobile, search, and ads cloaking, with 95.5% accuracy and a false positive rate of 0.9%.
- We measure the most prominent search and ad cloaking techniques in the wild; we find 4.9% of ads and 11.7% of search results cloak against Google's generic crawler.
- We determine the minimum set of capabilities required of security crawlers to contend with cloaking today.

II. Background & Related Work

We begin by outlining the incentives that bad actors have to conceal their webpages from crawlers. We also summarize existing techniques that websites employ to distinguish between crawlers and organic traffic. For the purposes of our study, we consider websites that deliver optimized content to small screens or localized visitors to be benign—our focus is exclusively on *blackhat cloaking*.

A. Web Cloaking Incentives

Web cloaking refers to the set of techniques that a web server uses to fingerprint incoming visitors in order to customize page content. Benign examples include servers that redirect mobile clients to pages optimized for small screens (e.g., m.nytimes.com) as opposed to content-rich desktop equivalents. The more insidious variety involves serving entirely distinct content to (security) crawlers in order to inflate a page's search ranking to drive traffic, evade ad quality scanners, or stealthily deliver drive-by exploits only to vulnerable clients. These techniques create a discrepancy in how the web is observed by bots and how it is perceived by organic clients.

There are many areas where blackhat cloaking can be beneficial for miscreants, but here we focus on the following three categories: search results, advertisements and drive-by downloads.

Search results: Cloaking is one tool in an arsenal of techniques that miscreants use for Search Engine Optimization (SEO). Servers will manipulate fake or compromised pages to appear enticing to crawlers while organic visitors are shepherded to (illegal) profit-generating content such as storefronts selling counterfeit luxury products, pharmaceuticals, and dietary supplements [16], [31], [32].

Advertisements: As an alternative to duping crawlers for free exposure, miscreants will pay advertising networks to display their URLs. Miscreants rely on cloaking to evade ad policies that strictly prohibit dietary scams, trademark infringing goods, or any form of deceptive advertisements–including malware [9], [36]. Ad scanners see a benign page while organic visitors land on pages hosting scams and malware. Time of check versus time of use (e.g., delayed URL maliciousness) may also play into a miscreant's cloaking strategy.

Drive-by downloads: Miscreants compromise popular websites and laden the pages with drive-by exploits. In order to evade security crawlers like Wepawet or Safe Browsing that visit pages with vulnerable browsers [6], [26], these payloads will first fingerprint a client and only attack vulnerable, organic visitors. While we include this for completeness, we focus our study on search and ad cloaking.

B. Prevalence of Cloaking

Previous studies have shown that finding instances of cloaking in the wild requires intimate knowledge of the search keywords or the vulnerable pages that miscreants target. Wang et al. estimated only 2.2% of Google searches for trending keywords contained cloaked results [32]. A broader method for finding cloaked URLs involved targeted searches for specific cocktails of terms such as "viagra 50mg canada" where 61% of search results contained some form of cloaking [32]. Leontiadis et al. reported a complementary finding where 32% of searches for pharmaceutical keywords advertised in spam emails led to cloaked content [16]. Keyword targeting extends to other realms of fraud: Wang et al. found 29.5% of search results for cheap luxury products contained URLs that redirected visitors to a cloaked storefront [31]. Our strategy for selecting URLs to crawl is built on top of these previous findings in order to minimize the bandwidth wasted fetching benign content.

C. *Cloaking Detection Techniques*

Researchers have responded to the steady adaptation of cloaking techniques over time with a medley of anti-cloaking (or de-cloaking) techniques. Early approaches by Wang et al. relied on a cross-view comparison between search results fetched by a browser configured like a crawler and a second fetch via a browser that emulated a real user [33], [34]. They classified a page as cloaking if the redirect chain deviated across fetches. This same approach dominates subsequent cloaking detection strategies that fetch pages via multiple browser profiles to examine divergent redirects (including JavaScript, 30X, and meta-refresh tags) [16], [17], [32], inspect the redirection chains that lead to poisoned search results [18], isolate variations in content between two fetches (e.g., topics, URLs, page structure) [31], [32], [35], or apply cloaking detection to alternative domains such as spammed forum URLs [24]. Other approaches exclusively target compromised webservers and identify clusters of URLs all with trending keywords that are otherwise irrelevant to other content hosted on the domain [14]. Our study improves upon these prior detection strategies. To wit, we build an anti-cloaking pipeline that addresses previously unexplored cloaking techniques as gleaned from the underground; and we explore additional approaches for cross-view comparisons that contends with page dynamism, interstitial ads, network-based cloaking, and the absence of divergent redirect paths. Our pipeline also improves on prior work as it discerns adversarial cloaking from geo targeting and content optimization for small screens. It does so by comparing across views both in terms of textual topic and entities detected in images. These improvements allow us to measure the dominant cloaking strategies in the wild, and in turn, inform search, ad, and malware pipelines that must contend with web cloaking.

III. Underground Perspective of Cloaking

Cloaking—like the commoditization of exploits, proxies, and hosting [2], [28]—is an infrastructure component for sale within the underground. To identify potential cloaking services and software, we first exhaustively crawled and indexed a selection of underground forums. We then ranked forum discussions based on the frequency of keywords related to cloaking software. The most discussed cloaking package was mentioned 623 times, with the author of the software engaging with the community to provide support and advertise new deals. The least popular service was mentioned only 2 times. Through manual analysis and labor, we successfully obtained the top ten most popular cloaking packages which ranged in price from \$167 for entry level products up to \$13,188 for the most sophisticated advertised features. We note that for legal protection and to avoid any potential de-anonymization of our underground identities, we cannot publicly disclose the names of the underground forums we harvested, or the names of the cloaking software under analysis.

We analyzed all ten cloaking packages in order to gain an insight into (1) fingerprinting capabilities; (2) switch logic for displaying targeted content; and (3) other built-in SEO capabilities such as content spinning and keyword stuffing. We use this tear down later in Section IV to design an anti-cloaking system capable of defeating all of the cloaking techniques we discover. We make no claim our investigation exhaustively covers cloaking software or whether our ten particular programs are widely used by miscreants. Indeed, the price and skill set required to operate some of the packages ensures their exclusivity to only the most affluent or sophisticated miscreants. That said, in conducting our analysis, we observed that the set of cloaking techniques among the packages we analyzed quickly converged to a fixed set of signals. This may indicate that while our coverage of cloaking software may be incomplete, the best cloaking techniques are frequently shared (or copied) much like exploit kits.

TABLE I: Cloaking fingerprinting capabilities reverse engineered from the most sophisticated six samples of cloaking software.

Capability	**Cloaking Type**	C_1	C_2	C_3	C_4	C_5	C_6
IP Address	Network	✓	✓	✓	✓	✓	✓
rDNS	Network	–	✓	✓	✓	–	✓
Geolocation	Network	✓	–	✓	✓	–	✓
User-Agent	Browser	✓	✓	✓	✓	✓	✓
JavaScript	Browser	✓	–	✓	–	–	✓
Flash	Browser	–	–	–	–	–	–
HTTP `Referer`	Context	✓	✓	✓	✓	✓	✓
Keywords	Context	✓	–	✓	–	–	–
Time window	Context	–	–	✓	–	✓	–
Order of operations	Context	–	–	✓	–	–	✓

A. *Cloaking Software Analysis*

Of the cloaking applications we analyzed, only one (coincidentally, the most expensive) protected itself with a tool with no publicly available unpacker. Languages for the various cloaking applications ranged from C++, Perl, JavaScript, and PHP. Sophistication ranged from drop-in scripts and plugins for Wordpress pages, while others included a custom compilation of Nginx for serving cloaked content. For each of the applications, we manually investigated the underlying cloaking logic and any embedded blacklists.

B. *Cloaking Techniques*

Cloaking techniques among the services we analyzed span a gamut of network, browser, and contextual fingerprinting. We present a detailed breakdown of key capabilities in Table I. These techniques only scratch the surface of browser fingerprinting that can otherwise enumerate screen size, font lists, header orders, and divergent JavaScript and HTML implementations [3], [7], [8], [20]–[23], [29]. Intuitively, cloaking services need only to deliver a fingerprinting technique to consumers that works. Without external pressure, there is no reason for cloaking developers to adapt toward more complex approaches proposed in research.

TABLE II: Breakdown of crawling bots covered by the blacklist-as-a-service. The top five search engines include Bing, Yahoo, Google, Baidu, and Yandex.

Crawler	Operator	Blacklisted IPs	Overall
msn.com	Bing	21,672	40.01%
yahoo.com	Yahoo	15,069	27.82%
googlebot.com	Google	5,398	9.97%
baidu.com	Baidu	2,846	5.25%
yandex.com	Yandex	1,092	2.02%
Other	Ask, Lycos, etc.	1,117	2.06%
Unknown	–	6,972	12.87%

TABLE III: Breakdown of the types of businesses targeted by the static blacklist, including the number of subnets and IP addresses in those ranges.

Entity Type	Distinct Entities	Subnets	IP Coverage
Hosting providers	42	508	12,079,768
Security companies	30	346	541,860
Internet service providers	27	49	1,419,600
Search companies	9	32	1,392,640
Other companies	3	12	34,628
Proxy networks	3	10	1,334
Academic institutions	2	14	17,106,682
Hacker collectives	2	4	60
Individuals	2	4	780
Registrars	2	4	19,136,508
Total	122	983	51,713,860

1) Network Fingerprinting

IP Address: Some crawlers make no effort to obfuscate the IP addresses ranges they appear from. This allows cloaking services to enumerate the set of bot IPs. Of the ten cloaking services we examine, four embed an IP blacklist; the others allowed operators to upload their own IP blacklist. Of the four IP blacklists, three mirrored the same blacklist-as-a-service available for a $350 annual fee. The list, updated twice daily, contained 54,166 unique IP addresses tied to popular search engines and crawlers at the time of our analysis. This same service provided a capability for cloaking clients to report back all IP addresses and HTTP headers tied to incoming visitors to a centralized server which the blacklist service recommended for timely detection of new crawling activities. We note that, despite us crawling sites that we later learned via side-channels relied on this blacklist, our IPs were never identified (potentially due to limited crawling or ineffective detection on the service's part).

Examining the blacklist-as-a-service in detail, we find that reverse DNS queries on the IP addresses surface crawlers from Bing, Yahoo, Google, Baidu, and Yandex as detailed in Table II—the top five search engines based on Alexa ranking [1]. A tiny fraction of IPs relate to Ask, Lycos, and smaller search engines. Consequently, any attempt to de-cloak content from these IPs—even with seemingly organic browser profiles—will fail. Another 6,972 IPs (12.9%) have no rDNS information and appear from a distinct /16 CIDR block than the top five search engines (which operate out of 77 CIDR /16 blocks). We examine whether any of these overlap with contemporaneous lists of proxies including 1,095 Tor exit nodes and 28,794 HideMyAss IPs. We find no evidence of Tor or HideMyAss blacklisting.

The last of the four blacklists contained a significant static list of CIDR blocks encompassing 51,713,860 IP addresses from 122 business entities as annotated in the blacklist (shown in Table III). The coverage advertised by the blacklist includes 30 security and anti-virus companies (e.g., Avira, Comodo, Kaspersky) as well as 9 popular search engines (e.g., Google, Microsoft, Yahoo). The list also covers multiple hosting providers, public clouds (e.g., Amazon), registrars, and proxy networks such as TOR that are unlikely sources of organic traffic. We also find CIDR blocks that blacklist entire ISPs, which as the blacklist author annotates, serve the crawlers for some security companies. Finally, the list contains a few academic institutions (e.g., MIT, Rutgers), hacker collectives (e.g., Germany's Chaos Computer Club), and individual researchers. We note that the abnormally large number of IP addresses associated with academic institutions results from universities such as MIT controlling an entire Class A subnet that the blacklist owner opted to cover.

Interestingly, when comparing the blacklist-as-a-service and the static list of IPs, we find only two subnets in common, both of which belong to Google. This indicates that the blacklist-as-a-service is geared toward targeting search engines for SEO, whereas the second blacklist focuses on security companies and researchers.

Reverse DNS: In the event a crawler appears from a non-blacklisted IP, four of the ten cloaking services perform a rDNS lookup of a visitor's IP. In the absence of a `NXDOMAIN` error, the software compares the rDNS record against a list of domains substrings belonging to Google (*1e100, google*), Microsoft, Yahoo, Baidu, Yandex, Ask, Rambler, DirectHit, and Teoma. Some of the cloaking services in turn add newly identified crawler IPs to their embedded blacklists. As such, any anti-cloaking pipeline must at a minimum crawl from IPs with non-overlapping (or completely absent) rDNS information.

Geolocation: We find that four of the ten cloaking services allow geographic targeting at a country level granularity based on mappings. One goes so far as to embed a duplicate of the MaxMind public GeoIP list for live querying. We do not observe any pre-configured list of blocked geo origins; all targeting is left to the software's operator. This poses a significant challenge to anti-cloaking pipelines as network infrastructure must support arbitrary network vantage points around the globe. However, as we previously discussed, most cloaking services fail to block Tor or major proxy providers. As such, we consider these services as a potentially acceptable sources of IP diversity.

TABLE IV: User-Agent substrings used by cloaking software to identify crawlers.

Crawler User-Agent substrings					
altavista	aol	askj	baidu	bingbot	crawler
gigablast	google	jeeves	lycos	msn	slurp
sogou	spider	teoma	yahoo	yandex	

2) Browser Fingerprinting

User-Agent: Well-behaving search and advertisement crawlers announce their presence with specialized User-Agent strings commonly advertised on the operator's website. Examples include Google's *googlebot*; Microsoft's *bingbot* and *msnbot*; and Yahoo's *slurp*. We find cloaking services ubiquitously rely on User-Agent comparisons to explicitly block Google, Bing, Yahoo, Ask, Baidu, Teoma, and Yandex. The cloaking services also carve out generic exceptions for *crawler* and *spider* to trigger cloaking logic. We provide a detailed breakdown of the exact substring matching patterns in Table IV. Some substrings capture overly broad applications. One cloaking product contends with this fact by including a whitelist covering *code.google.com/appengine* (third-party services operating on Google's infrastructure) and *via translate.google.com* (incoming translation requests likely from users); however, all other services make no exception.

JavaScript & Flash: JavaScript and Flash (or the lack thereof) can serve as both a crawler fingerprinting technique as well as a redirection delivery method. We find three cloaking services rely on JavaScript execution as a technique for blocking rudimentary crawlers. We find no support for Flash-based cloaking, though there is evidence of such attacks in the past [9]. One service also allows for operators to input custom JavaScript fingerprinting logic executed on each visit—a potential route to configure SEO-focused cloakers for drive-by exploit targeting, though we argue such attacks are orthogonal and more likely to come via exploit kits [12]. For our study, we opt to support both JavaScript and Flash when crawling to cover all possible bases.

3) Contextual Fingerprinting

HTTP `Referer`: The final ubiquitous cloaking technique we observe across blackhat applications involves scanning the HTTP `Referer` of incoming visitors to verify users originate from search portals. The default whitelist matches major crawlers (previously discussed in Table IV), though miscreants can disable this feature. This technique prevents crawlers from harvesting URLs and visiting them outside the context they first appeared. We contend with this cloaking approach by always spoofing a HTTP `Referer`, the details of which we expand on in Section IV.

Incoming Keywords: Keyword cloaking—supported by two services—takes HTTP `Referer` cloaking a step further and checks the validity of the search keywords that brought a purported visitor to a miscreant's page. Mechanistically, the cloaking software extracts the list of search terms embedded in the HTTP `Referer` using a set of customized regular expressions for each search portal and then compares the terms against an operator-provided list of negative keywords and positive keywords (e.g., viagra, Tiffany) associated with a page. The software triggers cloaking logic if a HTTP `Referer` contains any negative keywords or lacks any positive keywords. We adapt to this technique by embedding expected keywords in our HTTP `Referer`—derived from ad targeting criteria or page content. We note that since Google has been serving search results over TLS to logged-in users since 2013, and therefore not passing keywords in the HTTP referrer, the effectiveness of keyword cloaking has diminished. We have found that keyword cloaking is no longer pivotal, as it has been in previous works [32].

Time Window: Timing-based cloaking prohibits visitors from accessing uncloaked content more than once in a specific time window. The two services we find offering this capability rely on server side logs of a visitor's IP address. Any repeat visitors within 24 hours are redirected to cloaked content as if they were a crawler. This raises a potential issue of false negatives for an anti-cloaking pipeline without a sufficiently large IP pool.

Order of Operations: While most of the cloaking services we study rely on a single *doorway* page through which all cloaking logic triggers, two of the services supports multiple hops. Upon visiting a page, the software sets a short lived cookie (e.g., seconds). When legitimate users interact with the page, such as clicking on a URL, the next doorway checks whether this cookie is present and within the expiration window. This allows cloaked websites to enforce a specific sequence of actions (e.g., visit, click) where crawlers would likely enqueue the URL for visiting at a later time or on an entirely different machine. Our pipeline consolidates all crawling of a domain to a single short window.

C. Redirection Techniques

We observe three techniques that cloaking applications rely on to deliver split-view content upon successfully fingerprinting a client. The first involves redirecting clients via meta-refresh, JavaScript, or `30X` codes. Crawlers are either stopped at a *doorway* or redirected to an entirely different domain than legitimate clients. Alternatively, websites dynamically render content via server-side logic to include new page elements (e.g., embedding an iframe) without changing the URL that appears in a browser's address bar. The last technique involves serving only crawlers a `40X` or `50X` error. We account for each of these techniques when designing our anti-cloaking pipeline.

D. Built-in SEO Services

All but one (C_6) of the cloaking software under analysis supports automatic content generation aimed at SEO either natively or through third-party plugins (e.g., SEnuke, XRumer which are content spinning programs that create topically related documents [37]). The most advanced built-in solution takes a set of targeted search terms from the software's operator, after which the program will automatically query popular search engines for related content, scrape top-ranked links, and synthesize the content into a seemingly relevant document from a crawler's perspective. Furthermore, these systems automatically detect and exclude copyright infringing content and trademarked names to avoid penalization or removal by search ranking algorithms. As part of this document construction, the software will intersperse targeted search terms with stolen content to increase the frequency of related terms. We rely on this observation later in Section IV in order to detect contextual cloaking that requires a visitor's HTTP `Referer` to contain specific keywords as previously discussed in this section.

IV. Detecting Cloaking

We leverage our tear-down of blackhat cloaking techniques to build a scalable anti-cloaking pipeline that detects when two or more distinct browsers are shown divergent content. We preview our system's design in Figure 1. We start by aggregating a diverse sample of URLs to scan for cloaking (❶). We then fetch each of these URLs via multiple browser profiles as well as network vantage points to trigger any cloaking logic (❷). We finally compare the content, structure, and redirect graph associated with each fetch (❸) before feeding these features into a classifier to detect the presence of blackhat cloaking (❹). Whereas we note multiple prior studies have proposed techniques to de-cloak URLs in specific settings—particularly redirection cloaking—our goal with this system is to understand which anti-cloaking techniques *generalize* across web cloaking (including mobile and reverse-proxy cloaking), and similarly, to understand the *minimum* set of capabilities required of security scanners to contend with the current cloaking arms race. We defer concrete implementation details till Section V.

A. Candidate URL Selection

To conduct our study we aggregate a stream of URLs from popular websites, search results, and mobile-targeted advertisements. We split our dataset into two: one part for training a classifier based on labeled data from previous studies of cloaking outlined in Table V; and a second sample that we feed into our classifier to analyze an unbiased perspective of cloaking in the wild shown in Table VI.

Our training corpus of benign URLs consists of a random sample of pages appearing in the Alexa Top Million, all of which we assume to be non-cloaked.[1] We later validate this assumption in Section VII. For labeled instances of cloaked domains we rely on a feed of counterfeit luxury storefronts that fingerprint Google's search bot, maintained by Wang et al. [31], collected between February, 2015–May, 2015. In total, we rely on 94,946 URLs for training. We note our labeled dataset has a class imbalance toward non-cloaked content which helps to emphasize false negatives over false positives.

TABLE V: Breakdown of labeled data we use for training and evaluating our classifier.

Labeled Dataset	Source	Volume
Legitimate websites	*Alexa*	75,079
Cloaked storefronts	*SEO abuse [31]*	19,867
Total		94,946

TABLE VI: Breakdown of unlabeled data that we classify to study cloaking in the wild.

Unlabeled Dataset	Source	Volume
Luxury storefronts	*Google Search*	115,071
Health, software ads	*Google Ads*	20,506
Total		135,577

For studying blackhat fingerprinting techniques, we rely on a sample of unlabeled URLs tied to Google Search and Google Ads targeting mobile users.[2] Unlike our training corpus, these carry no biases from previous de-cloaking systems. Due to the targeted nature of cloaking as previously explored by Wang et al. [32], we predicate our searches on high-value keywords related to luxury products (e.g., gucci, prada, nike, abercrombie) and ad selection on keywords related to weight loss (e.g., garcinia, keatone, acai) and popular mobile software applications (e.g., whatsapp, mobogenie). While it is possible this targeting biases our evaluation, we argue that the content that miscreants serve is independent from the underlying technology used to fingerprint crawlers. In total, we collect 135,577 URL samples, only a fraction of which we assume will actually cloak.

B. Browser Configuration

While our dataset may appear small, this is because we crawl each URL with 11 distinct browser and network configurations in an attempt to trigger any cloaking logic, repeating each crawl three times to rule out noise introduced by dynamic content or network errors. In total, we perform over 7 million crawls. We detail each crawl configuration in Table VII. We chose this set based on our domain knowledge of reversed blackhat services which target various platforms, environment variables, JavaScript, Flash, and cookie functionality. Also, these configurations cover the three possible vantage points of anti-cloaking deployments: a search engine advertising its crawlers with their User Agents and IP addresses, a browser farm deployed in the cloud, and a stealthy deployment using mobile and residential networks. In section VI-D, we evaluate how each of these vantage points contribute to our overall cloaking detection performance.

In total, we provide three native platforms for fetching content: Chrome on Desktop; Chrome on Android; and a

[1]We reiterate that we treat personalization, geo targeting, and reactive design as benign and thus non-cloaking. Many of these techniques are present in the Alexa Top Million. We use a cloaking label only for blackhat techniques.

[2]We assume that Google deploys some defenses against cloaking. As such, our dataset will capture only mature cloaking techniques that evade immediate detection by Google crawlers.

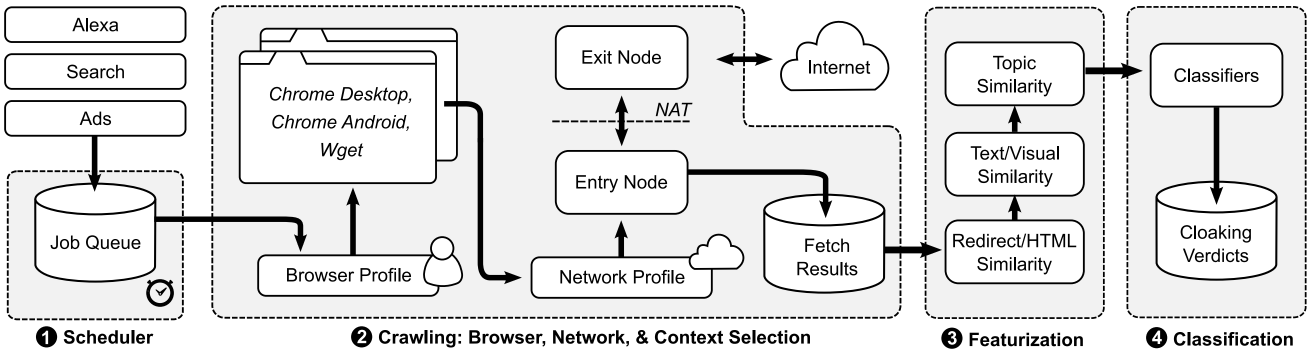


Fig. 1: Cloaking detection pipeline. We crawl URLs from the Alexa Top Million, Google Search, and Google Ads (❶). We dispatch requests to a farm of stratified browser profiles covering mobile, desktop, and simple crawlers and network configurations that encompass residential, mobile, and cloud IP addresses to fetch content (❷). We then compare the similarity of content returned by each crawl (❸), feeding the resulting metrics into a classifier that detects divergent content indicative of cloaking (❹).

TABLE VII: List of browser, network, and contextual configurations supported by our system.

Profile Name	Platform	User-Agent	Network	Referrer	Click
Googlebot Basic	HTTP Request only	Googlebot	Google	✗	✗
Googlebot Desktop	Chrome Desktop	Googlebot	Google	✗	✓
Googlebot Android	Chrome Android	Googlebot	Google	✗	✓
Basic Cloud (no referer)	HTTP Request only	Chrome OSX	Cloud	✗	✗
Basic Cloud	HTTP Request only	Chrome OSX	Cloud	✓	✗
Chrome Desktop Cloud (no referer)	Chrome Desktop	Chrome OSX	Cloud	✗	✓
Chrome Desktop Cloud	Chrome Desktop	Chrome OSX	Cloud	✓	✓
Chrome Mobile Cloud (no referer)	Chrome Android	Chrome Android 4.4	Cloud	✗	✓
Chrome Mobile	Chrome Android	Chrome Android 4.4	Cloud	✓	✓
Desktop User	Chrome Desktop	Chrome OSX	Residential	✓	✓
Mobile User	Chrome Android	Chrome Android 4.4	Mobile	✓	✓

basic HTTP fetch that supports cookies and `30X` redirects, but does not handle Flash, JavaScript, meta redirects, or embedded content (e.g., `iframes`, images). We then configure the User-Agent of each platform to mirror the latest version of Chrome on Mac OSX; Chrome on a Nexus 5 Android device; or the Google search bot. Finally, we wire the browser to proxy all requests through a pre-defined network (e.g., mobile, cloud) described shortly. After a successful fetch, we save both the HTML content of a page along with a screenshot. Due to the possibility of URL visits introducing browser state, we tear down our environment and clear all cookies between fetches. As we show later in Section VII, in practice only a few of these profiles are necessary to detect all cloaked content (though not measure the precise cloaking logic). Security crawlers can adopt this slimmed-down configuration to increase overall throughput.

Context Selection: We support spoofing three contextual features when fetching URLs: the keywords a user searches for; the path our crawler takes to reach a destination (e.g., the HTTP `Referer`); and user actions taken upon reaching a page (e.g., clicking). To determine which keywords to spoof, we first fetch every non ad-based URL with a basic Googlebot absent any HTTP `Referer` and then extract the (stuffed) keywords on the page. Methodologically, we filter a page's HTML to include only visible, non-stopword text, after which we select the top three most frequent words. Due to how miscreants spin content to achieve a high search rank (as discussed in Section III), these keywords are identical to those miscreants expect from legitimate clients. For ad-based URLs, the process is simpler: we rely on the keywords the advertiser bids on for targeting (gathered from Google AdWords).

Since browsers have complex policies on when to set the referrer (depending on whether the source and destination URLs are over TLS, and the type of redirect [30], [33]), we have opted not to spoof a crawler's path by simply overwriting the `Referer` field, as the difference in referrer handling might not trigger the uncloaking of a target website. Instead, we first load a Google search page and explicitly create a new element on the page via JavaScript that directs to the destination URL along with the aforementioned keywords embedded in the URL's parameters, after which we click the element. We support this approach for all types of URLs.

Finally, to handle click walls that appear due to order of operation-style cloaking, upon reaching a URL's final landing page we wait a few seconds and then select the largest element and simulate a user click event. If the click fails to cause the browser to load a different URL, we ignore the element and repeat the process until a loading event occurs or no elements are left to consider. In this fashion we create a realistic context in which a user visits a search engine, enters an appropriate search query, clicks on the cloaked page's search, ad, or drive-by URL; and ultimately interacts with the cloaked page's content. Note that we only click when using Chrome Desktop or Mobile, and not when using the simple HTTP fetcher (per Googlebot's typical behavior).

Network Selection: We proxy network requests through a tap that records all inbound and outbound traffic. Additionally, this tap provides a configurable exit IP belonging to either Google's network; a mobile gateway in the United States belonging to AT&T or Verizon; a datacenter network belonging to Google Cloud; or a residential IP addresses belonging to the researchers involved in this project. As some of our mobile and residential IPs exist behind NATs with dynamically allocated IP addresses, we establish a reverse tunnel to an entry point with a statically allocated IP address that forwards requests on to the exit node. Our diverse IP address pool allows us to evade or trigger network-based cloaking on demand.

C. Features

Deviations in page content introduced by cloaking include entirely unique HTML, distinct link locations, alternate cross-origin content, or only a new button appearing on a page. We compare the textual, visual, topical, and structural similarity of content between all possible pairs of browser configurations (e.g., Googlebot, Mobile User). Given we crawl every URL three times per candidate profile, we heuristically select the fetch that generated the largest volume of HTTP logs to serve as the representative sample. We make no assumptions on how significantly documents must differ to constitute blackhat cloaking. Instead, we rely on classification to learn an optimal cut that differentiates divergent content due solely to blackhat cloaking versus benign dynamism (e.g., breaking news, mobile optimization).

1) Pairwise Similarity Features

Content Similarity: We detect cloaking that returns entirely distinct content by estimating the similarity of each document's visible text and overall HTML. We begin by removing all whitespace and non-alphanumeric content (e.g., punctuation, formatting). We then tokenize the content using a sliding window that generates successive ngrams of four characters. Finally, we calculate a 64-bit simhash of all of the tokens which converts high-dimensional text into a low-dimensional representation amenable to quick comparison, originally pioneered for de-duping documents in search indexes [5]. To measure the similarity of two documents, we calculate the Hamming distance between two simhashes which is proportional to the number of document tokens that failed to match. A high score indicates two documents differ significantly. We run this calculation twice, once for only paragraph and heading text (that is, visible text in the page) and again for all HTML content.

Screenshot Similarity: Our second similarity score estimates visual differences in the layout and media presented to browsing profiles of the same window dimensions (e.g., Mobile, Desktop). This approach helps cover syntactically insignificant HTML changes, such as introducing an `iframe`, that significantly change a page's visible content. During each crawl, we take a screenshot both upon visiting a URL and after clicking on the largest hotlinked element. We convert these screenshots from RGB into a grayscale $n \times m$ array, after which we normalize each cell's pixel intensity to a range $[0, 255]$. We then calculate the per-pixel differences between two screenshots S_1 and S_2, opting to measure the total pixels that differ rather than the absolute value of their difference:

$$diff = \sum_{x=0}^{n} \sum_{y=0}^{m} S_{x_1,y_1} \neq S_{x_2,y_2} \tag{1}$$

A high score in this regard indicates a substantially different visual layout.

Element Similarity: While textual differences between crawls may arise due to dynamic advertisements or newly generated comments, deviations between a website's template should be less likely. To capture this intuition, we extract the set of URIs E associated with all embedded images per document. We then calculate the difference in media content between two documents by using the Jaccard similarity coefficient:

$$1 - \frac{|E_1 \cap E_2|}{|E_1 \cup E_2|} \tag{2}$$

A high score indicates there were multiple embedded images absent from one or another crawl. To measure the similarity in the page HTML structure, we repeat this same process with `divs` and `iframes`, first stripping the elements of any attributes, and then calculating the fraction of overlapping tags as an additional measure of structural similarity.

Request Tree Similarity: We compare the network requests generated while crawling to detect divergent redirects, mismatched network errors, and additional content. We begin by representing a sequence of network events (e.g., `GET`, `POST` requests) as $E = \{e_1, e_2, \ldots e_n\}$ where an event e_i consists of a tuple $\langle$*Method, Domain, Response Code, Path*$\rangle$. For two sequences E_1 and E_2 of potentially variable length, we calculate the number of differing requests independent of any timing information using the Jaccard similarity calculation previously outlined in Equation 2. A high score indicates that crawls triggered divergent network behavior. As an extension, we separately calculate the difference in total response packet size between two browsing profiles.

Topic Similarity: As an alternative to the previous fine-grained similarity metrics which may suffer in the presence of dynamic website content, we compare the overall semantic similarity of webpage content by extracting representative topics based on visible text. Mechanistically, we rely on an Latent Dirichlet allocation (LDA) implementation to extract a set of at most ten topics T per document [10], [13]. We then calculate the similarity between two document's topics T_1, T_2, repeating the Jaccard index calculation presented in Equation 2. A high score indicates the topics between pages differs significantly.

Screenshot Topic Similarity: Finally, due to potentially heavy reliance on media rather than text, we also compare the topic similarity of documents as detected by a deep convolutional neural network that uses screenshots as an input [15]. As with our text-based approach, for each screenshot we determine up to ten topics T that describe the visual content. We then repeat the same similarity calculation as outlined with text-based topics. We use this method to catch pages that display additional spam images (typically with sexual or pharmaceutical content) that drastically changes a page's perceived topic.

2) Per-page Dynamism Features

We estimate the natural, potentially legitimate dynamism of individual pages per browser configuration to help our classifier identify a minimum threshold above which divergent content is likely indicative of blackhat cloaking. As previously noted, we crawl each URL three times per browsing profile which we denote C_1, C_2, C_3 for clarity. We recompute all of the previous metrics for each possible pair C_i, C_j for $i \neq j$, averaging each metric to arrive at a single per-feature, per-page dynamism estimate. Finally, we provide the classifier both these similarity scores as well as the previous cross-browser pairwise similarity metrics divided by our dynamism estimates for that same feature (effectively calculating the ratio of cross-profile dynamism with intra-profile dynamism).

3) Domain-specific Features

We extend our feature set with domain-specific signals that target the entire collection of content crawled per URL (as opposed to pairwise metrics), outlined below. We use these for both classification as well as to simplify analysis by embedding meta-data about how miscreants cloak.

JavaScript, Meta, Flash Redirection: We include a single boolean feature for whether a server redirects our crawler via JavaScript, `meta-refresh` tag, or Flash to a domain that does not match the final landing page. Such behavior is common for (compromised) doorway pages where cloaking logic operates.

Googlebot Errors: We compare the size of requests returned to our basic Googlebot profile against all other profiles to determine whether a server provides the crawler an error. Several of the cloaking packages we reverse engineered offer an option of serving Googlebot a page aimed at downplaying the relevance of the page. Examples include a 404 interstitial (e.g., "this site is no longer available"), parked domain page, or fake security message such as "this site has been banned").

Landing Domains: We annotate each URL with the total number of landing page domains reached during all 33 visits, with an intuition that divergent landing sites are suspicious.

D. Classification

We employ Extremely Randomized Trees—an ensemble, non-linear, supervised learning model constructed from a collection of random forests where candidate features and thresholds are selected entirely at random [11]. For training, we rely on our labeled dataset of benign URLs from Alexa and cloaked search (discussed earlier in this section). Prior to classification, we normalize all features into a range $[0, 1]$ to simplify the interpretation of which signals are most salient. During classification, we rely on ten-fold cross validation. We discuss the overall performance of this classifier in Section V and its application to a holdout testing set for analysis in Section VII.

V. Implementation

We implement our system on Google Compute Engine with crawling and featurization distributed among 20 Ubuntu machines. The classification is performed on a single instance. Our scheduler is built on top of Celery backed by Redis. We compose crawling tasks as a tuple of a URL and profile that includes the target browser, network vantage point, and context to appear from. Celery handles distributing tasks to workers, monitoring success, and resubmitting failed jobs to live workers. We operate three types of crawlers: a basic robot that fetches URL content via the Python Requests library, akin to `wget`; a headless instantiation of Chrome controlled via Selenium, configured with a User-Agent for Mac OSX; and the same Chrome browser except in mobile emulation mode mimicking the Nexus 5 device with version 4.4 of the Android operating system. Our network vantage points include the authors' residential networks, Google's cloud network, Google's internal network, and mobile gateways belonging to AT&T and Verizon as purchased via pre-paid plans. We capture and log all network requests via mitmproxy. Finally, for featurization and classification we rely on scikit-learn [25], Pandas [19], and a mixture of libraries previously mentioned in Section IV for estimating content similarity and topic modeling.

VI. Evaluation

In this section, we explore the overall performance of our classifier, sources of false positives, the most salient features, and the feasibility of unsupervised clustering. To conduct our evaluation, we first train and test a decision tree classifier using 10-fold cross validation over our imbalanced dataset of 75,079 non-cloaked URLs and 19,867 cloaked URLs previously detailed in Section IV. We rely on a grid search to tune classifier parameters related to our decision forest (e.g., number of trees, depth of trees), ultimately electing the configuration with the optimum overall accuracy.

A. Overall Supervised Performance

We present the overall accuracy of our system in Table VIII. We correctly detect 99.1% of Alexa URLs as non-cloaked with a false positive rate of 0.9%. To achieve this degree of accuracy, we overlook 18.0% of potentially cloaked counterfeit

TABLE VIII: Performance of the supervised-learning classifier on the labeled train/test set, with 10-fold stratified cross validation.

Accuracy	TN Rate	TP Rate	FN Rate	FP Rate
95.5%	99.1%	82.0%	18.0%	0.9%

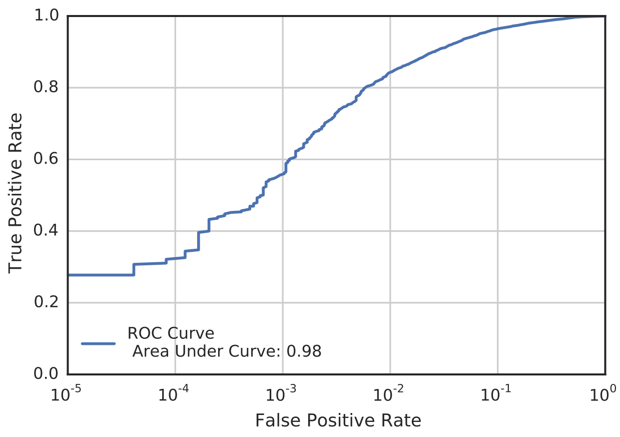


Fig. 2: Receiver operating characteristic curve for the supervised classifier (log-scale).

storefronts. If we examine the trade-off our system achieves between true positives and false positives, presented in Figure 2, we find no inflection point to serve as a clear optimum. As such, operators of de-cloaking pipelines must determine an acceptable level of false positives. For the remainder of our study we rely on a false positive rate of 0.9%.

B. Source of Errors

False Positives: We manually investigate a random sample of URLs our classifier mislabeled to understand the principle cause of errors. Qualitatively, we find three sources of false positives: (1) websites revising content between crawls, (2) connectivity issues, and (3) noisy labels where some Alexa URLs in fact cloak. In our current crawler implementation, we fail to enforce a time window during which all crawls must complete. This raises the risk that content substantially changes between successive fetches, incorrectly triggering our detection. We can solve this problem moving forward by enforcing an SLA on time-to-crawl. A similar problem arises if our crawler receives a `40X` error or if a page is not fully loaded when we take a screenshot, resulting in divergent image-based and network-based similarity scores. Along this vein, we also find instances where CloudFlare DDoS protection automatically blocks a fraction of our crawls, instead displaying an interstitial "checking your browser" which we mistake for a malicious interstitial. Finally, in rare cases, we find that some of the top Alexa sites serve cloaked ads that swap content when presenting to a crawler, likely unbeknownst to the site embedding the ad. These instances, as observed from our classifier, are in fact true positives, thus our overall false positive rate will be lower in practice.

False Negatives: We execute a similar qualitative analysis for false negatives, finding the majority of errors arise due to stability issues tied to cloaked websites. In particular, while crawling pages multiple times from the same profile we often find that the storefronts involved will throw transient errors. This causes intra-profile similarity metrics to deviate as strongly as cross-profile metrics, preventing an accurate assessment. We investigate whether cloaking servers introduce these errors intentionally (e.g., potentially blacklisting our crawler's IP address), but we find no correlation between a successful crawl and repeated errors afterward. Instead, errors appear randomly during any crawl. Similarly, we find some of the URLs were taken down or expired by the time we crawled them. These pages return the same error regardless of profile, thus leading the classifier to believe they are not cloaking. We also find a high volume of counterfeit storefronts that do not cloak, indicating that our labeled dataset is noisier than expected from a cloaking perspective (while its original collection was to study scams). These latter two sources of errors indicate that our false negative rate is likely lower in practice, though non-negligible.

Comparison to Previous Work: We note that our definition of cloaking is the most comprehensive to date, including mobile cloaking, graphical-only cloaking, and testing advanced cloaking techniques such as rDNS cloaking. Because of this, the performance of our classifier is not comparable with previous works that target specific types of cloaking, such as redirection cloaking [18], [27] or referrer and user agent cloaking [32]. If we restrict our detection to a specific type of cloaking, such as redirection cloaking, our classifier exhibits low to no false positives. However, such technique-specific restrictions yield a low recall that overlook sophisticated cloaking types such as when cloaking software replaces a single embedded image's content to deliver a cloaked ad offering. As our study aims to give a comprehensive overview of cloaking techniques in the wild, we opted to favor high recall at the expense of some false positives.

Additionally, our work is the first to distinguish between benign cloaking (e.g., mobile optimization, geolocalization, personalized results) from blackhat cloaking. For example, our detector is capable of determining that the mobile and desktop version of cnn.com differ in content, but that difference results exclusively from content optimization and should not be labeled as blackhat cloaking. This challenge leads to a higher degree of false negatives as we favor precision over recall to reduce false positives from polluting our subsequent analysis.

C. Salient Features

We rank the importance of the top 20 features that impact the accuracy of our classifier according to their Gini importance [4], effectively calculating the weight of the feature across all trees in our decision forest. We present our findings in Figure 3. The classifier associates the highest weight with JavaScript redirects that cross an origin boundary. Indeed, 41.8% of labeled cloaking URLs rely on this technique compared to 0.62% of presumed non-cloaking URLs. The remaining top 19 features span a spectrum of feature categories covering content, network request, topic, and screenshot similarity among multiple combinations of browsing profiles. This indicates all of our similarity metrics and profiles contribute to classification results, each covering a non-overlapping approach to cloaking.

Exploring feature salience in more detail, we examine the overall accuracy of our system when trained only a single class of similarity metrics. Figure 4 indicates that comparing the structure of pages is the most effective technique for minimizing false negatives, whereas topic, embedded element, and screenshot-based similarity metrics perform the worst in isolation. We recast this same measurement in Figure 5, except this time removing only a single comparison method from training. We find that a model with no screenshot similarity introduces the most false negatives, while removing page structure alone has the least impact. Our findings reiterate that an ensemble of comparison techniques are required to accurately detect split-view content.

D. Minimum Profile Set

Finally, we quantify the trade-off between anti-cloaking pipeline performance, and its efficiency and complexity. To do so, we start with the full system described here, and we repeatedly identify the crawling profile that, when removed, least impacts the false positive rate. The result of this greedy search of the anti-cloaking pipeline with the minimum capabilities is shown in Figure 6.

As with all classification scores shown here, the scores are the mean values in a ten-fold stratified cross validation. The results indicate that an anti-cloaking pipeline would still have an acceptable performance without a mobile user on a mobile network, and without the content similarity feature class. If any more capabilities are subtracted, the false negative rate doubles, whereas the false positive rate remains fairly low even for a pipeline composed only by a mobile browser, desktop browser and Googlebot, all crawling from Google IPs and cloud IPs. These browsers support clicking, taking screenshots, and visit URLs with the same profile repeatedly. Since any further simplification of this basic anti-cloaking pipeline doubles the false positive rate, we have established that this is the minimum anti-cloaking platform that is both efficient, by avoiding unnecessary crawling and featurization, and effective against current cloaking. We caution readers that this evaluation of the minimum viable anti-cloaking pipeline should be performed routinely, so to react in a timely manner to a spread in popularity of more advanced cloaking techniques.

E. Unsupervised Alternative

Supervised learning requires a steady stream of training data in the event miscreants adapt their techniques for displaying split-view content. As a potential alternative, we compare our supervised classifier's accuracy to that of an unsupervised clustering equivalent based on Gaussian mixture models. We measure accuracy based on the cluster's separation of our labeled dataset. We present our results in Table IX. Unsupervised learning achieves an overall accuracy 84.6% and false positive rate of 9.7% compared to supervised learning which achieves an accuracy of 95.5% with 0.9% false positives. While this indicates there is substantial power in merely comparing the similarity of content between multiple clients, a supervised classifier far outperforms clustering when labeling edge cases.

TABLE IX: Performance of the unsupervised-learning classifier on the labeled train/test set.

Accuracy	TN Rate	TP Rate	FN Rate	FP Rate
84.6%	90.3%	61.1%	38.9%	9.7%

TABLE X: Prevalence of cloaking in Google Search and Ads for URLs tied to high-risk keywords.

Source	Keyword Category	% Cloaking
Google Ads	Health, software ads	4.9%
Google Search	Luxury storefronts	11.7%

VII. Cloaking in the Wild

Having vetted our classifier, we apply it to an unlabeled dataset of 135,577 search results and advertisements targeting high-value, commonly abused keywords related to luxury products, weight loss, and mobile gaming. We measure the prevalence of cloaking in the wild and categorize the blackhat techniques involved.

A. Frequency

In Table X, we show the incidence of cloaked content for Google Search results and Google Ads. We estimate 4.9% of mobile-targeted advertisements predicated on high-risk keywords direct to cloaked content. This demonstrates that miscreants readily attempt to abuse the advertisement ecosystem as a distribution channel for nutraceuticals and knock-off mobile games and justifies a heightened scrutiny for specific ad targeting practices. We make no claim that Google fails to detect these cloaked URLs; only that miscreants purposefully provide evasive content to Googlebot while URLs remain operational.[3] For search, we find 11.7% of URLs in the top 100 results direct to cloaked content. This is consistent with previous estimates of pharmaceutical-based cloaking in Google search results where Wang et al. estimated 9.4% of search results linked to cloaked doorways [32]. Our results illustrate that cloaking remains a common blackhat practice requiring constant maintenance on the part of security crawlers to keep pace with the arms race. That said, our measurements show that miscreants more commonly target search results over advertisements, likely due to the cost of advertising.

[3]We do not track whether URLs are eventually pulled, precluding any longitudinal evaluation for how long it takes before ads are disabled.

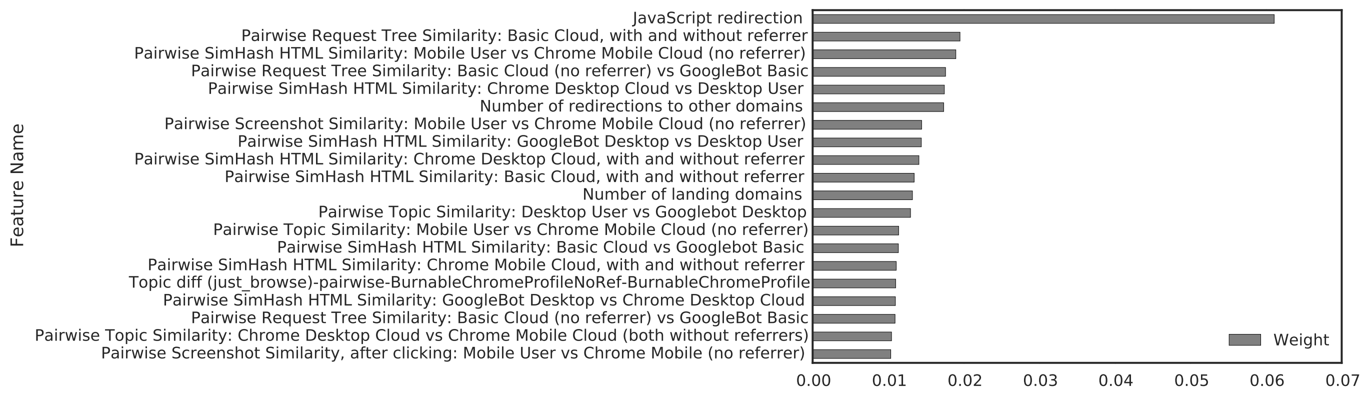


Fig. 3: Top 20 features selected by our classifier ranked by their weight in our model.

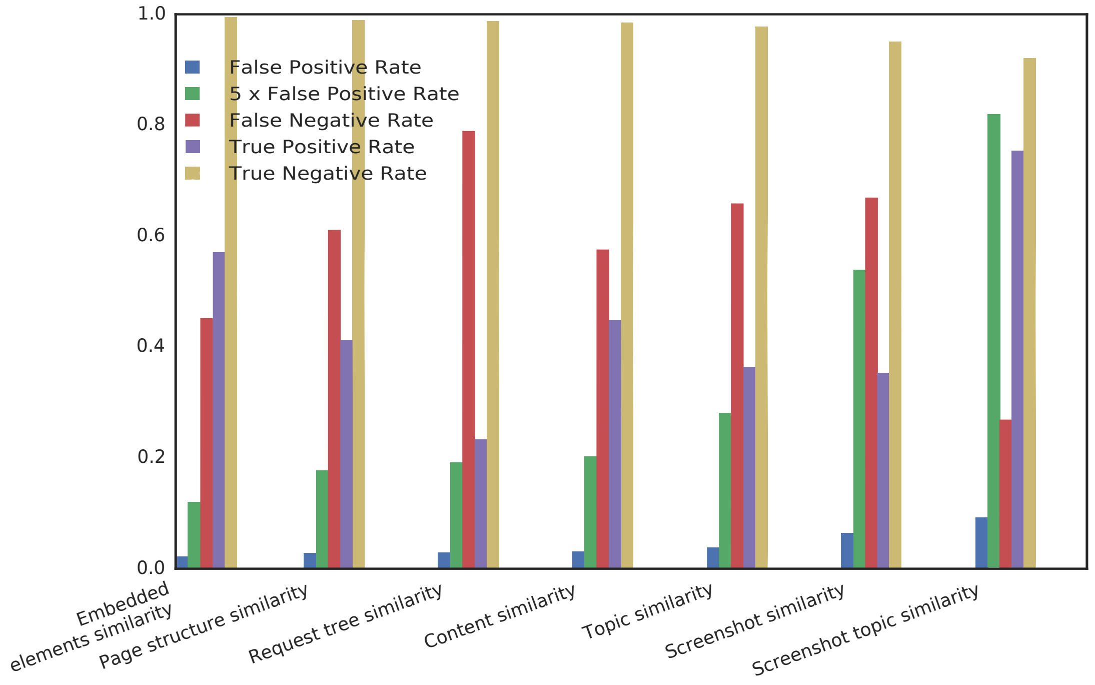


Fig. 4: Classifier performance when training only a single feature class. We include a magnified false positive rate ($5x$) to emphasize an otherwise minimal variation. We order feature classes by ascending false positive rates.

B. Targeting Techniques

Cloaking sites hide their payload from everyone but the intended audience of organic users. We analyze how miscreants arrive at this distinction and study any differences between ad and search-based cloaking. To do so, first we mark all cloaking URLs in the unlabeled dataset with our full classifier. Then, for each class of targeting techniques, such as checking that the visitor has a HTTP `Referer` set, we train a classifier on our labeled dataset but specifically exclude browser profiles that include the targeting technique under evaluation. We then measure the fraction of cloaking URLs in the unlabeled dataset that this new classifier identifies as cloaking, effectively acting as a proxy for which targeting criteria is critical to receive decloaked content (and thus accurate detection). Note that we use the unlabeled dataset as our test set to mitigate any bias in our labeled dataset.

We show the fingerprinting checks miscreants use for Google Search and Google Ads in Table XI. We find the most prominent targeting criteria is the presence of JavaScript which miscreants use for 49.6% of cloaked ads and 22.4% of cloaked search results. This is followed in popularity by checking for Googlebot's IP address and User-Agent, and finally evidence that a client interacts with a page (e.g., clicking). Our results highlight that any anti-cloaking pipeline must come outfitted with each of these capabilities to accurately contend with cloaking.

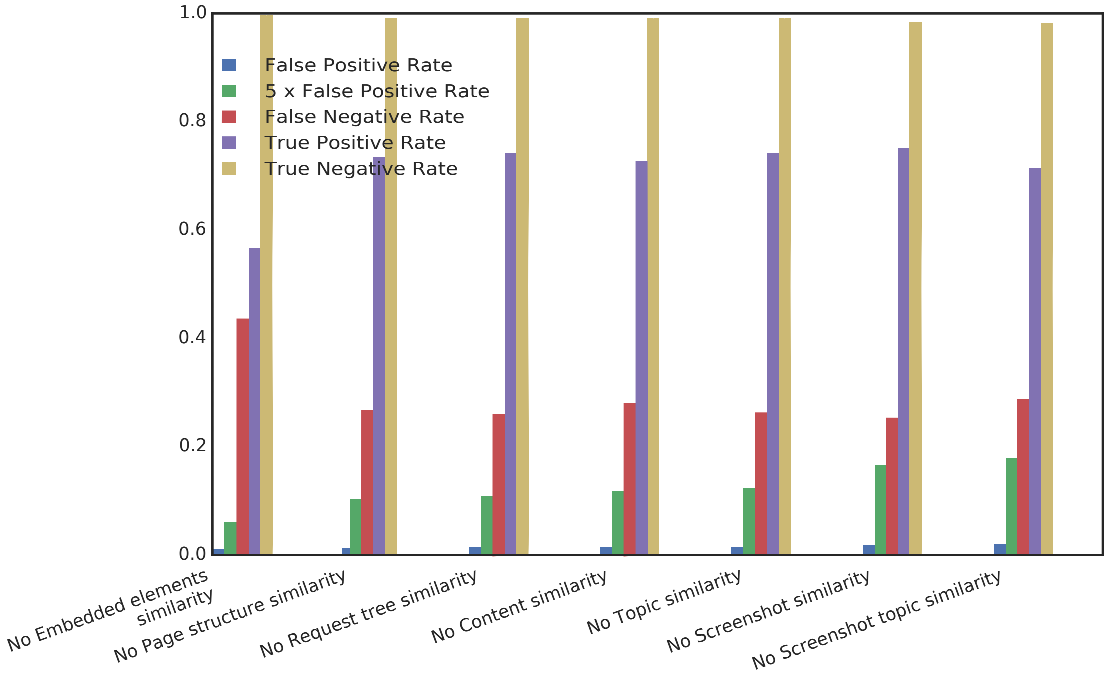


Fig. 5: Classifier performance when training on all but one class of features. We include a magnified false positive rate ($5x$) to emphasize an otherwise minimal variation. We order feature classes by ascending false positive rates.

TABLE XI: Fingerprinting techniques in the wild that are used to make a cloaking decision. Broken down for Google Search and Ads.

Fingerprinting check	*Google Search*	*Google Ads*
Has referrer set?	6.1%	5.4%
User has clicked?	10.6%	18.0%
Is Google (IP, User Agent)?	14.3%	20.7%
Has JavaScript support?	22.4%	49.6%
Is mobile device?	4.9%	8.5%

TABLE XII: Delivery techniques in the wild, broken down for Google Search and Ads. Same-page modifications include server-side targeting as well as client-side rendering.

Cloaking Type	*Google Search*	*Google Ads*
30X redirections	33.6%	19.9%
40X client errors	12.0%	8.5%
50X server errors	2.5%	4.4%
JavaScript redirections	29.9%	6.6%
Same-page modifications	22.0%	60.6%

C. *Delivery Techniques*

Cloaking sites deliver their uncloaked content to organic visitors in a variety of ways. Some sites opt to redirect visitors to a monetized URL, either via a server-side decision (via a 30X redirection), or on the client-side via JavaScript. To be less conspicuous, other sites opt to display the uncloaked content directly in the landing page, either through a reverse proxy, or a modification of the DOM such as adding `div`, `img`, or `iframe` elements. We analyze the most popular delivery techniques in our dataset as determined by our network logs for sites labeled as cloaking, broken down by type in Table XII. We find delivery techniques in the wild differ substantially between search results and advertisements. For instance, JavaScript redirects account for 29.9% of cloaked search URLs compared to 6.6% of ads, with ads instead favoring same-page modifications. Our result highlights that while miscreants may cloak against products using a variety of techniques, our anti-cloaking system nevertheless succeeds at generalizing and captures each approach. Any security crawler must address each of these techniques as well as prepare for future iterations of the cloaking arms race.

VIII. Case Studies

In this section we present case studies exemplary of the monetization strategies among the Google Search and Google Ads URLs we identified as cloaking.

Lead Generation for Mobile Apps: We encountered multiple sites that entice mobile users to install both dubious and legitimate third-party apps. Interestingly, a minority of Alexa's top domains also exhibit this behavior. For example, we show how mobile and desktop visitors see opensubtitles.org in Figure 7. When this site detects a visitor with an Android mobile User-Agent and a HTTP referrer set, it adds a new *div* element via JavaScript. This element randomly loads an ad for a legitimate Android app, or is stylized as fake Android notification. When clicked, this notification leads to a dubious app that acts as a free app store. When installed, this app riddles the device with unwanted ads (through the AirPush library).

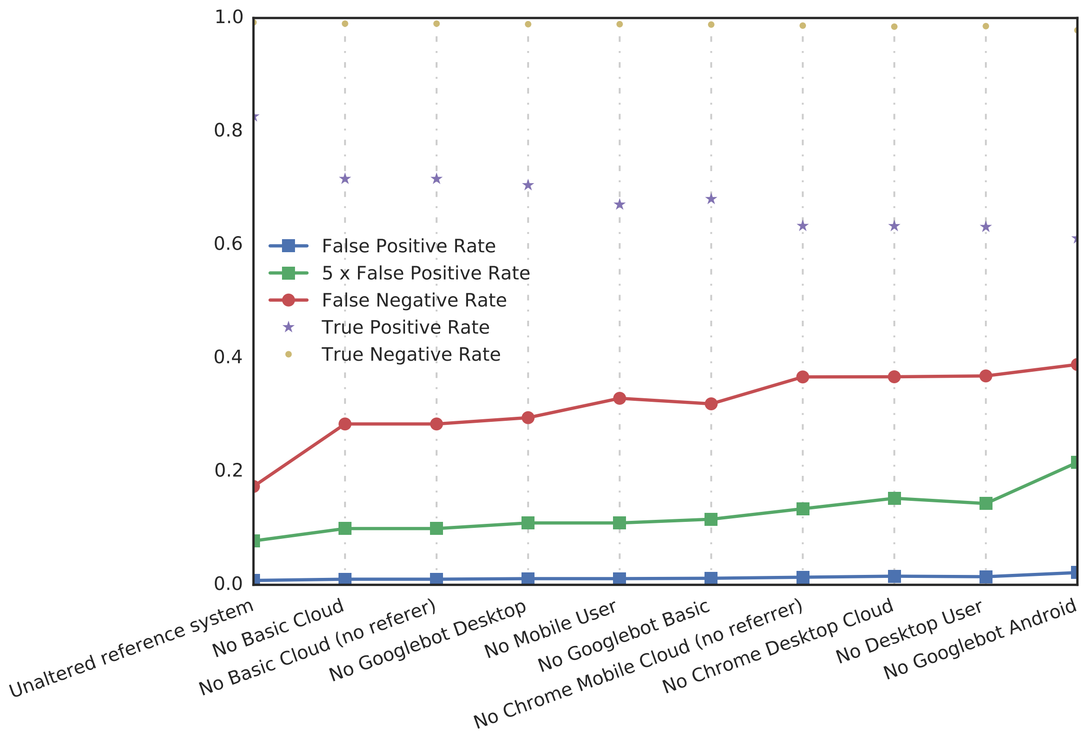
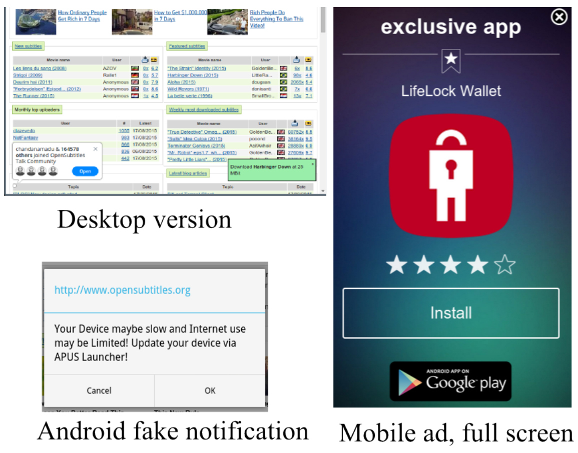


Fig. 6: Classifier performance degradation, when repeatedly removing the crawling profile that least affects the false positive rate.

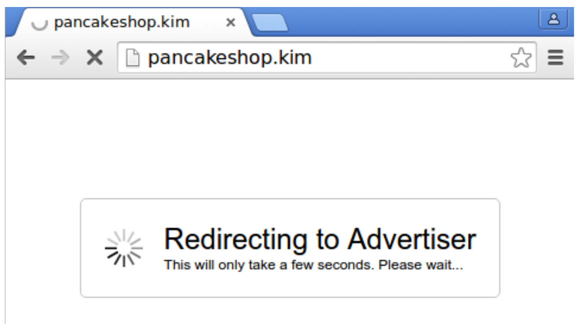


http://www.opensubtitles.org

Your Device maybe slow and Internet use may be Limited! Update your device via APUS Launcher!

Cancel OK

Fig. 7: Observing opensubtitiles.org from different devices can yield completely different content.



Redirecting to Advertiser

This will only take a few seconds. Please wait...

Fig. 8: Cloaking site that redirects to advertisements.

Malware: A few of the cloaking websites we identified are distributing malware. For example, *saomin.com*, delivers to mobile user an Android app that is flagged as malicious by 19 AntiVirus engines on VirusTotal. In another case, the user was encouraged to install a malicious browser extension called *FromDocToPDF*.

Traffic Resellers: We have also observed cloaking sites selling their organic traffic to a ring of advertisers. For example, in Figure 8 we show a screenshot of *pancakeshop.kim*. This site redirects users to third-party advertisers based on the type of platform and *Referer* header. Also, this site geolocates the visitor and uses this information to decide which ads to run. Visiting the site from outside the US yields a blank page, or a message “We currently don’t have any sponsors for this domain name”.

Some traffic reseller employ a wide set of rules to decide what content to display. An example of this is macauwinner.tk, which pretends to be a parked domain when visited from outside the US, whereas it delivers tailored content to users on residential and mobile networks, detecting their Internet provider (e.g., it also displays “Dear AT&T Uverse user”). Also, it delivers content specific to the operating system of the user, mimicking its appearance when creating fake windows and alert boxes. The pages to which users get redirected range from fake AntiViruses, to fake popular websites (e.g., Facebook), to surveys.

Affiliate fraud: We found cases where the cloaking site performs affiliate fraud. For example, *drseks.com*, redirects every other user to a major shopping retailer with an affiliate id set. By doing so, this retailer shares a fraction of the profits from a sale to the cloaked domain.

IX. Breaking the Cloaking Arms Race

As miscreants adopt increasingly sophisticated application-specific cloaking techniques, it becomes difficult for defenders to keep pace with the cloaking arms race. Currently, our system is a viable solution, as it is designed to defeat current cloaking capabilities. We have determined the minimum capabilities a current anti-cloaking pipeline would need precisely to guide the design of such a pipeline, spending engineering time efficiently. On the long run, however, we envision that miscreants will add to their cloaking arsenal (e.g., carrier-specific mobile cloaking), increasing the cost of detection at the expense of driving less organic traffic to their concealed offers. To counter this trend, we propose two possible alternatives that would render it significantly harder for miscreants to deliver split-view content, although they would require an in-browser component.

Client-side Cloaking Detection: As cloaking hinges on serving benign content to search engine and ad network crawlers, one option is for those same services to embed a succinct digest of a webpage's content in the parameters tied to search and advertisement URLs. When users are redirected after clicking on one of these URLs, the user's browser can compare the newly served content against the crawler's digest. If the two substantially differ, the browser can raise a warning interstitial that alerts the user to a suspected scam, phishing, or malware attack. Mechanistically, this comparison naturally follows the pairwise features we laid out for our anti-cloaking system. The benefit over our current architecture is that crawlers no longer need to maintain multiple browsing profiles or network vantages—clients provide the second view. Additionally, this approach respects the privacy of the users, as only the potentially-dangerous pages will be reported by the participating (i.e., opted-in) users.

There are however some open challenges with this approach. First, dynamic content remains a concern. If miscreants can limit the deviations introduced by cloaking to within typical norms (e.g., including only a small new button or URL), the system may fail to detect the attack. That said, this also constrains an attacker in a way that reduces click through from users. Additionally, there is a risk with news sites and other frequently updated pages that a crawler will serve incoming visitors a stale digest due to an outdated crawl, thus burdening users with alerts that are in fact false positives. To avoid this, the crawler would either need to immediately re-crawl the page to confirm the change and suppress the alert, or the digest should account for the category of the site, allowing for a higher threshold for news sites.

Distributed Client Content Reporting: To overcome the problem of staleness, we consider an alternative model where a user's browser opts to anonymously report a content digest after clicking on a search result or advertisement to the associated search engine or ad network. This server would then review the incoming digest against the copy fetched by its crawler. In the event of a mismatch, the server would immediately re-crawl the URL to rule out the possibility of an outdated digest. If there is still a client-server mismatch after crawling, the search engine or ad network involved could pull the reported URL from public listing to protect all future clients. From a privacy perspective, the server receiving reports would already be aware the user clicked on the URL, such as how search engines currently redirect visitors through analytic interstitials. However, as users may click through to a signed-in page containing sensitive content (e.g., facebook.com), the digest reported must not leak personalized content. Furthermore, this approach opens servers up to an abuse problem where malicious clients may spoof digests to unduly trigger the removal of legitimate search results and advertisements. However, assuming there are more legitimate clients than malicious and some form of rate limiting, servers can rely on majority voting to solve this problem, though the long tail of URLs may yet pose a challenge.

X. Conclusion

In this work, we explored the cloaking arms race playing out between security crawlers and miscreants seeking to monetize search engines and ad networks via counterfeit storefronts and malicious advertisements. While a wealth of prior work exists in the area of understanding the prevalence of content hidden from prying eyes with specific cloaking techniques or the underlying monetization strategies, none marries both an underground and empirical perspective that arrives at precisely how cloaking operates in the wild today. We addressed this gap, developing an anti-cloaking system that covers a spectrum of browser, network, and contextual blackhat targeting techniques that we used to determine the minimum crawling capabilities required to contend with cloaking today.

We informed our system's design by directly engaging with blackmarket specialists selling cloaking software and services to obtain ten of the most sophisticated offerings. The built-in capabilities of these packages included blacklisting clients based on their IP addresses, reverse DNS, User-Agent, HTTP headers, and the order of actions a client takes upon visiting a miscreant's webpage. We overcame each of these techniques by fetching suspected cloaking URLs from multiple crawlers that each emulated increasingly sophisticated legitimate user behavior. We compared and classified the content returned for 94,946 labeled URLs, arriving at a system that accurately detected cloaking 95.5% of the time with a false positive rate of 0.9%.

When we deployed our crawler into the wild to scan 135,577 unknown URLs, we found 11.7% of the top 100 search results related to luxury products and 4.9% of advertisements targeting weight loss and mobile applications cloaked against

Googlebot. In the process, we exposed a gap between current blackhat practices and the broader set of fingerprinting techniques known within the research community which may yet be deployed. As such, we discussed future directions for breaking the cloaking arms race that included clients reporting browsing perspective to crawler operators, hindering the ability of miscreants to show benign content exclusively to search engines and ad networks.

References

[1] Alexa. Alexa top 500 global sites. http://www.alexa.com/topsites, 2012.

[2] Ross Anderson, Chris Barton, Rainer Böhme, Richard Clayton, Michel J.G. van Eeten, Michael Levi, Tyler Moore, and Stefan Savage. Measuring the cost of cybercrime. In *Proceedings of the Workshop on Economics of Information Security (WEIS)*, 2012.

[3] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Information Security Technology for Applications*, pages 31–46. Springer, 2012.

[4] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[5] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[6] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *Proceedings of the 19th International Conference on World Wide Web*, 2010.

[7] Peter Eckersley. How Unique Is Your Web Browser? In *Privacy Enhancing Technologies (PET)*, 2010.

[8] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, 2015.

[9] Sean Ford, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Analyzing and detecting malicious flash advertisements. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, 2009.

[10] gensim. models.ldamodel – Latent Dirichlet Allocation. https://radimrehurek.com/gensim/models/ldamodel.html, 2015.

[11] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[12] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J. Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, D. McCoy, Antonio Nappa, Andreas Pitsillidis, et al. Manufacturing compromise: The emergence of exploit-as-a-service. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[13] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *Neural Information Processing Systems*, 2010.

[14] John P John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martín Abadi. deseo: Combating search-result poisoning. In *Proceedings of the USENIX Security Symposium*, 2011.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Nektarios Leontiadis, Tyler Moore, and Nicolas Christin. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. In *USENIX Security Symposium*, 2011.

[17] Nektarios Leontiadis, Tyler Moore, and Nicolas Christin. A nearly four-year longitudinal study of search-engine poisoning. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[18] Long Lu, Roberto Perdisci, and Wenke Lee. Surf: detecting and measuring search poisoning. In *Proceedings of the 18th ACM conference on Computer and communications security*, 2011.

[19] Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th*, volume 445, pages 51–56, 2010.

[20] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in javascript implementations. In *Proceedings of the Workshop on Web 2.0 Security and Privacy*, 2011.

[21] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. In *Proceedings of the Workshop on Web 2.0 Security and Privacy*, 2012.

[22] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, Edgar Weippl, and FC Wien. Fast and reliable browser identification with javascript engine fingerprinting. In *Proceedings of the Workshop on Web 2.0 Security and Privacy*, 2013.

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

[24] Yuan Niu, Hao Chen, Francis Hsu, Yi-Min Wang, and Ming Ma. A quantitative study of forum spamming using context-based analysis. In *NDSS*. Citeseer, 2007.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, and et al. Weiss. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iFRAMEs point to us. In *Proceedings of the 17th Usenix Security Symposium*, pages 1–15, July 2008.

[27] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 133–144. ACM, 2013.

[28] Kurt Thomas, Danny Yuxing Huang, David Wang, Elie Bursztein, Chris Grier, Thomas J. Holt, Christopher Kruegel, Damon McCoy, Stefan Savage, and Giovanni Vigna. Framing dependencies introduced by underground commoditization. In *Proceedings of the Workshop on the Economics of Information Security*, 2015.

[29] Thomas Unger, Martin Mulazzani, Dominik Fruhwirt, Markus Huber, Sebastian Schrittwieser, and Edgar Weippl. Shpf: enhancing http(s) session security with browser fingerprinting. In *Proceedings of the International Conference on Availability, Reliability and Security*, 2013.

[30] W3C. Referrer Policy. http://w3c.github.io/webappsec/specs/referrer-policy/, 2015.

[31] David Y Wang, Matthew Der, Mohammad Karami, Lawrence Saul, Damon McCoy, Stefan Savage, and Geoffrey M Voelker. Search+ seizure: The effectiveness of interventions on seo campaigns. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014.

[32] David Y Wang, Stefan Savage, and Geoffrey M Voelker. Cloak and dagger: dynamics of web search cloaking. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2011.

[33] Yi-Min Wang and Ming Ma. Detecting stealth web pages that use click-through cloaking. In *Microsoft Research Technical Report, MSR-TR*, 2006.

[34] Y.M. Wang, M. Ma, Y. Niu, and H. Chen. Spam double-funnel: Connecting web spammers with advertisers. In *Proceedings of the International World Wide Web Conference*, pages 291–300, 2007.

[35] Baoning Wu and Brian D Davison. Detecting semantic cloaking on the web. In *Proceedings of the 15th international conference on World Wide Web*, 2006.

[36] Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014.

[37] Qing Zhang, David Y Wang, and Geoffrey M Voelker. Dspin: Detecting automatically spun content on the web. In *Symposium on Network and Distributed System Security (NDSS)*, 2014.