Web Runner 2049: Evaluating Third-Party Anti-bot Services

Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, Nick Nikiforakis

Web Runner 2049: Evaluating Third-Party Anti-bot Services

Babak Amin Azad[1], Oleksii Starov[2], Pierre Laperdrix[3], and Nick Nikiforakis[1]

[1] Stony Brook University
[2] Palo Alto Networks
[3] CNRS / Univ. Lille / Inria

Abstract. Given the ever-increasing number of malicious bots scouring the web, many websites are turning to specialized services that advertise their ability to detect bots and block them. In this paper, we investigate the design and implementation details of commercial anti-bot services in an effort to understand how they operate and whether they can effectively identify and block malicious bots in practice. We analyze the JavaScript code which their clients need to include in their websites and perform a set of gray box and black box analyses of their proprietary back-end logic, by simulating bots utilizing well-known automation tools and popular browsers.
On the positive side, our results show that by relying on browser fingerprinting, more than 75% of protected websites in our dataset, successfully defend against attacks by basic bots built with Python scripts or PhantomJS. At the same time, by using less popular browsers in terms of automation (e.g., Safari on Mac and Chrome on Android) attackers can successfully bypass the protection of up to 82% of protected websites.
Our findings show that the majority of protected websites are prone to bot attacks and the existing anti-bot solutions cannot substantially limit the ability of determined attackers. We have responsibly disclosed our findings with the anti-bot service providers.

1 Introduction

The modern web is home to benign and useful bots, such as, search engine crawlers that provide easy access to information around the web. Yet the same technology that enables benign bots is also utilized by malicious bots which disrupt services, steal business and customer information, and make illicit profits for their operators. Malicious bots are used to automatically find and exploit vulnerabilities on websites (such as outdated and vulnerable Content Management Systems) [15], scrape email addresses and content for sending spam and creating phishing websites, registering thousands of accounts and selling them via underground markets (e.g. for fake followers on social networks [47]) and brute forcing login forms with credentials stolen from other websites (known as *credential stuffing*). Some of the recent bot attacks include, ride-sharing companies scraping pricing and vehicle information from their competitors websites [9]. Similarly, the bots targeted the airline industry, causing an increase in the look-to-book ratio which leads to increased fees [11].

According to recent estimates, more than 50% of traffic on the web belongs to bots with more than half of that belonging to malicious ones [19]. In this environment, specialized anti-bot services have emerged which offer bot detection and bot blocking as a service to their clients. Even though these services claim to utilize an impressive array of technologies, their operation and effectiveness in detecting and blocking bots have not been evaluated.

In this paper, we report on the first analysis of 15 popular anti-bot services. We identify the JavaScript code which their clients deploy on their websites and perform a white box analysis of its operation. We observe heavy reliance on browser fingerprinting including recent fingerprinting techniques that fingerprint a system's graphics card, local IP address, and even whether the browser attempts to lie about its identity. To understand whether this extracted information is sufficient to detect and block abusive bots, we utilize six different existing automation tools, ranging from off-the-shelf crawlers, to automated browsers. Through the use of carefully designed experiments, we evaluate the ability of the most popular anti-bot services to stop attacks, such as, content scraping, credential brute forcing, and account hijacking.

Among others, we find that few services are capable of significantly slowing down attackers and that certain unusual crawling tools, such as, an AppleScript-controlled Safari Browser and an ADB-controlled Android smartphone can successfully crawl large numbers of webpages and conduct account attacks. More specifically, at least 68% of our simulated scraping requests were not blocked, and more than 90% of our account takeover attempts were successful with at least one of the tested tools. In addition, for more than half of our target websites, there is at least one tool that enables us to do 1,000 password brute force attempts without getting blocked.

Contrary to our expectations, we discover that having a bot reach websites from a public cloud does not significantly decrease its performance since existing services put more emphasis on browser fingerprints rather than source IP address.

2 Background

Since malicious bots can lie about their identity, prior research has proposed a number of methods for bot detection, including behavior-based detection (based on the premise that bots browse websites differently than real users [24, 29]), detection based on accessing content that is invisible for regular users [39, 52] and more recently, based on browser fingerprinting [14]. Once a visitor is suspected to be a bot, the website can request the solving of CAPTCHAs, rate-limit the user, or altogether block traffic from the offending IP address.

Even though web developers may try to implement the aforementioned techniques, it is unlikely that the developers of small websites can keep up with the adaptation from the side of the bot authors. In order to keep up with attackers, new businesses have emerged that sell bot-detection services to their clients, similar to anti-DDoS companies which protect their clients against DDoS attacks. Website owners can then integrate these services with their website to block bots without needing to know how a bot was identified. One major benefit of using such services compared to a custom implementation of an anti-bot mechanism, is the threat-information sharing that happens in the background. If bot activity with certain characteristics is detected on website A, website B that is also a client of the same anti-bot service, can get information about this bot and block it immediately at its first interaction.

Anti-bot companies advertise a range of bot-related attacks which they can detect and stop. By analyzing the descriptions of their services, we summarize the attack scenarios performed by malicious bots into the three following categories (these attacks are discussed in more detail in Section 5):

- **Account Takeover**, also known as *credential stuffing*, refers to automated login attempts with stolen or leaked credentials to target websites. In this case, attackers may take

advantage of users reusing credentials across services and leaked password databases found in underground markets.

– **Credential Brute Force** is another type of account takeover attack. In this scenario, the attacker uses a list of popular passwords against user accounts to break into them.
– **Content Scraping** is an automated attempt to steal proprietary website information, such as product price lists and inventory, to gain a business advantage.

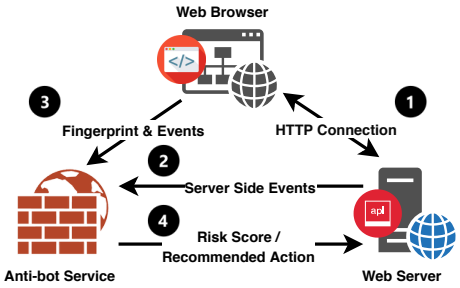

Fig. 1: *High-level architecture of Anti-bot services*

The general structure for anti-bot services is depicted in Figure 1. We arrived at this architecture by studying the design of multiple anti-bot services and abstracting away service-specific details. When users visit a website protected by an anti-bot service, fingerprinting scripts gather information from their browser and send it directly to the anti-bot service's back-end. Information about how users interact with the website and actions taken, such as, login attempts and viewed pages, are also transmitted to the anti-bot service by the webserver using server-side APIs. Plugins are typically provided for popular content management systems (such as WordPress and Drupal) and integration is also available at the website and webserver layers using provided SDKs. In this architecture, every visitor of the website has a unique identifier which is later used by the webserver to query the anti-bot service and receive a risk score. Depending on the configuration of the website, different thresholds on the risk score can trigger different events, such as, showing a CAPTCHA, limiting the number of requests of suspicious users, or altogether blocking them.

A key component of each service is their *fingerprinting scripts*, which attempt to collect as many signals as possible for distinguishing between human and bot-like behavior. Browser fingerprinting has evolved substantially in the past few years from querying simple JavaScript APIs [20] to the rendering of complex 3D scenes with WebGL [16]. By collecting a range of information about the browser, the operating system and the hardware of a device, anti-bot services can obtain a precise view of the overall browsing system which can be used for detecting bots [14]. Next, such services usually claim to have sophisticated machine learning models on their back-end servers, which are trained to identify bot-related fingerprints on large volumes of data that they observe across their clients. In order to get a complete view of these services, both the coverage of fingerprinting features, as well as the accuracy of their back-end models have to be measured to quantify their effectiveness. Hence, in this study, we perform an analysis of their deployed fingerprinting scripts (Section 3), as well as gray box and black box testing of anti-bot back-end models (Section 5). Our experiments allow us to not only

Table 1: *Popular anti-bot services*

Service	Type	# Clients	Alexa 1M
Cloudflare (G)	Bot attacks	7,250,835	13.65%
Sift Science	Bot attacks	18,733	3.41%
Iovation	Account fraud	14,280	1.62%
ShieldSquare (G)	Bot attacks	8,151	1.46%
PerimeterX	Bot attacks	7,808	1.14%
InfiSecure	Bot attacks	5,443	0.11%
DataDome (G)	Bot attacks	912	5.48%
ThreatMetrix	Account fraud	628	5.41%
Distil Bot Defense	Bot attacks	484	38.43%
Castle (G)	Account fraud	260	4.62%
Simility	Account fraud	182	2.20%
ThisData	Account fraud	138	1.45%
Kount Access	Account fraud	124	31.45%
Unbotify	Bot attacks	60	3.33%
DupZapper	Account fraud	33	3.03%
Overall	Anti-bot	7,311,809	13.56%

capture the effectiveness of each anti-bot service in detecting bots, but to also measure how well websites interpret and act upon the risk score reported by each anti-bot service.

3 Analysis of anti-bot services

For our analysis, using popular search engines, we searched for phrases such as, "bot detection" and "bot prevention", and compiled a list of 15 popular services in September 2017. Table 1 lists the discovered services ranked according to their number of clients. The process of identifying client websites is described in Section 5.

Overall, we see that almost half of the anti-bot services have thousands of client websites with Cloudflare being the most popular service having 13.65% of its clients from the Alexa's top 1 million websites. The number of clients for Cloudflare in Table 1 represents the total number of websites observed using Cloudflare. The numbers are based on "BuiltWith" website statistics, which provides reports on web technologies [10]. Since all other services specialize only in bot protection, we already know that clients that use these services want to defend against bots. Whereas for Cloudflare, there can be various reasons to use their service such as DDOS protection, CDNs, or for adding HTTPS support to a website. Therefore, only a subset of these websites might configure Cloudflare's firewall to block bots.

Services in Table 1 marked with (G) indicate those for which we could acquire an account (trial or paid) without having to talk to a sales representative. For these services, we were able to conduct gray box testing, in addition to the black box testing for all services. Among the services we study, we can distinguish the following two main types:

– **Universal solutions against bot attacks** usually collect fingerprints and user-behavior data from clients using JavaScript and other common browser fingerprinting methods. They also collect information from the web server including the specific actions taken by users, such as, the browsing of a specific page, or the submission of a form.

– **Specific services against account fraud** that focus on the defense against account takeover and credential stuffing attacks. These services make use of both bot detection and anomalous account activity to identify attacks.

All of the anti-bot services listed in Table 1, except Cloudflare, use fingerprinting scripts on their clients' websites to assist them in bot detection. We collected client-side fingerprinting scripts from the 14 anti-bot services that use this technique. Next to beautifying and statically analyzing the JavaScript code, we dynamically executed the scripts in order to inspect the sent payloads and detect what fingerprinting-related APIs they utilize. For that, we used a custom browser extension (following the approach of Lerner et al. [28]) that can intercept browser API calls on a page.

3.1 Fingerprinting and automation detection mechanisms

First, we observed that most services use standard fingerprinting features, such as, screen properties, available fonts, plugins and MIME types. We observed similar features being collected to those reported by Vastel et al. [50]. The particular techniques in extracting these features differ, e.g. some services directly enumerate the `navigator.plugins` object, some simply use the PluginDetect library [7], and others have further custom checks. In comparison, we witness that few services incorporate more recent fingerprinting techniques, such as, Canvas or WebGL fingerprinting that can provide a more accurate view of the system's hardware.

Another finding, supporting the fact that anti-bot scripts attempt to capture obvious signs of web automation, is the variety of checks to detect PhantomJS, Nightmare [2], Selenium, and headless Chrome browsers. Different services use different techniques, such as, printing a stack trace and searching for the "selenium" keyword or probing for the existence of known variables (e.g., `window.callPhantom`). By deploying these checks against our own Selenium installations, we discovered that most of the deployed checks do not work for recent Selenium versions (except the `navigator.webdriver` property which is still present on the Selenium ChromeDriver).

Even though not all services use state-of-the-art fingerprinting techniques, those that do also try to detect inconsistencies in the collected browser fingerprints. For example, the user-agent sent by Selenium can be modified to look like a Firefox browser on Android, or Safari on iOS. The problem is that these modifications can lead to inconsistencies where modified and unmodified attributes cannot possibly belong to the same browsing environment. Three services make use of client-side code to detect such cases of mismatch between attributes.

Table 2: *Known fingerprinting libraries*

Source library	# Services	Source library	# Services	Source library	# Services
Fingerprintjs2 [4]	4	PluginDetect [7]	3	fonts2.swf [1]	1
FontList.swf [4]	3	Evercookies [3]	1	Modernizr [5]	1

As Table 2 shows, a significant number of anti-bot services rely on existing fingerprinting libraries, such as, the popular Fingerprintjs2 [4]. We also observed services that use other advanced fingerprinting features, including the detection of the local IP address through WebRTC and Flash, as well as the recording of user actions, i.e., mouse moves and clicks. Some collect and send this data only once, whereas others periodically collect and report this

information. Finally, we discovered a number of cases where more novel fingerprinting techniques were used, like the recently deprecated Battery status API [38] (Castle), AudioContext fingerprinting (PerimeterX), and even DOM changes to a supplied HTML page with different input fields (ThreatMetrix) which can be used to detect browser extensions [44]. This demonstrates that a small number of anti-bot services are closely following browser-fingerprinting research, and incorporate this research in their products.

A service which includes anti-bot functionality but differs from the rest is Cloudflare [17]. Unlike the evaluated third-party anti-bot services, Cloudflare itself is responsible for all resolutions of their clients' domains. As a result, Cloudflare can detect and block traffic at their servers, without any input from their clients. After analyzing the requests, we observed that Cloudflare does not perform any type of client-side fingerprinting using JavaScript or Flash. Cloudflare mostly relies on IP reputation (historical malicious activity), firewall rules based on HTTP requests, and rate limiting to prevent automated and malicious behavior.

3.2 Anti-bot service integration with websites

As depicted in Fig. 1, anti-bot services communicate the decision (often in the form of a risk score) to their clients upon each request. In this section we present our observations on how risk scores or decisions are communicated to clients and how the websites react to these reports.

Communicating the raw risk score. Services like Cloudflare, directly communicate the score to their clients and let them decide which thresholds to choose when blocking bots (e.g., show a CAPTCHA when risk score is greater than 50).

Communicating the final verdict. Services like Castle interpret the risk score internally and communicate the final verdict (Allow, Challenge, or Block) to the client websites through their API. Website administrators can then decide to show a CAPTCHA or notify the user via third-party channels.

Handling everything in the background. These services analyze the collected fingerprints and events, redirecting users to CAPTCHAs or block pages. As a result, the whole process of decision making happens in the background and website administrators have no control over it. Occasionally, there are no tunable parameters exposed to administrators which means that false positives have to be communicated and remediated through customer-support channels.

Finally, next to communicating the risk score and decision making, how websites react to bots is also defined by the anti-bot service. Some services have the ability to be deployed inline with the web traffic (e.g., Distil Bot Defense and Cloudflare). In this scenario they can straightforwardly redirect malicious users to CAPTCHAs and block pages. Similarly, the integrated SDK can communicate with the anti-bot service and redirect the detected threats to specific block pages. Lastly, the reaction may be left up to the website developers. In the example of Castle, website developers can decide to block the request or notify the users about the breach.

4 Available tools for building bots

In this section, we introduce the tools that we utilized to evaluate whether the anti-bot services are capable of detecting attackers of different levels of expertise (reflected in the complexity of their tools). We categorize the tools that are available for attackers in three groups, covering multiple levels of complexity:

- **Basic bots:** The least sophisticated approach is based on general-purpose automation tools (e.g., Python Requests and PhantomJS). *Python Requests* scripts are capable of sending GET and POST requests but do not execute JavaScript (these are conceptually similar to utilizing command-line tools, such as, `wget` and `curl`). This is the most basic approach that we expect to be detected by anti-bot services. We also include *PhantomJS* in this category, which was the first easy-to-script, headless, JavaScript-supporting browser [40] and therefore attracted a great deal of abuse [41].
- **Automated Browsers:** The second and more sophisticated category involves using real browsers (e.g., Firefox and Chrome) automated by Selenium. These bots can often be augmented with user-action simulation, such as, mouse moves, floating delays, and page scroll.
- **Less Popular fingerprints:** Anti-bot companies claim to share threat information between their clients. As a result, common tools used to create bots can be detected more effectively. Contrastingly, attackers can incorporate less popular tools to potentially bypass bot detection mechanisms, due to their limited history of malicious activity. To model this approach, we use AppleScript-automated Safari and ADB-automated Chrome on Android.

5 Experimental Setup

To analyze the effectiveness of anti-bot services in terms of preventing bot activity, we utilize a number of real-world attack scenarios. In this section, we describe the categories of our tests, and how we utilize tools from different bot categories presented in Section 4. We implement a large number of web automation scripts that can interact with websites at different levels of complexity. Each test is comprised of attack and tool combination and is executed from hosts with IP addresses belonging to our campus and a public cloud. These addresses are picked from a pool of 30 campus IP addresses and 30 cloud IP addresses distributed across 8 geographical regions.

5.1 Gray Box Experiments

For the companies that we could obtain paid or trial accounts, we integrate their SDK with our testing website (a WordPress-based, web application). Under the gray box scenario, we run our tests in a fully controlled environment where we control both the bots as well as the website receiving the bot traffic. By monitoring the administration panel provided by the anti-bot service, we have access to the final decisions to allow or block the traffic. Nevertheless, the machine learning models and decision boundaries used to classify the incoming traffic is still a black box. As such, we call these set of tests, gray box.

1. Test Preparation Initially, for each "attack category," and "web automation tool," we create scripts to mount the attacks and measure their success. Our tests cover the following categories:

- **Account Takeover:** In this setup, we create an account on websites utilizing anti-bot services (either our own for gray box testing or third-party websites for black-box testing) from a fixed geographical location, IP address, and browser. We then attempt to automatically login to this account from different geographical locations and IP addresses using our bots. This discrepancy in login location, browser fingerprint, and use of automation tools should, in principle, trigger the account-takeover protection system to prevent the "malicious" login activity or alert the user.

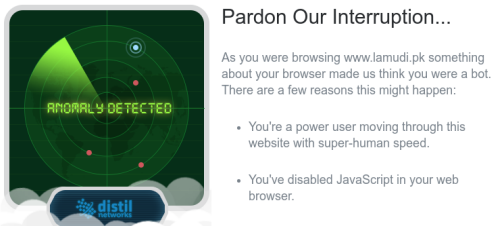

Fig. 2: *Screenshot of blocking message from Distil Networks*

- **Credential Brute Force:** To implement this scenario, we use our web automation tools and try to login with 1,000 invalid credentials. We then measure the number of requests before getting blocked. According to prior work [26,32], at least 4% of passwords created under different password policy schemes can be found in under 1,000 guesses.
- **Content Scraping:** By extracting product list and pricing information from 1,000 pages, we evaluate whether the anti-bot service will block our bots.

2. Test Execution Each attack script is executed from different IP addresses which we rotate as necessary. The hosts are located on our campus and on Linode (a public cloud provider). The reason for using two different locations is to simulate attackers who have access to premium IP address space that is not associated with crawling activity, versus attackers who can just rent virtual machines on public clouds. We do not perform multiple attacks at the same time to ensure that the detection of one attack does not affect the detection of another.

3. Post-processing of Results After each test, we inspect the anti-bot administration panel and look for reports of blocked bots based on the IP address we used for each attack.

5.2 Black Box Experiments

Most services require their potential customers to talk to a sales representative, prove their identity as a real business, and go through a series of interactions to acquire and adopt anti-bot services. Since we cannot perform gray box testing for these service, we devised a set of black box experiments.

Data Collection After compiling our list of anti-bot services that we wish to evaluate, we crawl the web and find websites that adopt these services. Starting with a list of known clients for each anti-bot service (e.g., list of clients on the website of anti-bot services), we analyze their websites to identify unique content, such as, JavaScript files or DOM elements that can be used as a signature to detect more clients of each service. The resulting signatures are then queried in the PublicWWW [8] and NerdyData [6] code-search engines and the results are supplemented with our own crawls of the Alexa top 1 million websites.

Given the number of tests we wish to conduct and that we need website-specific scripts that can fill forms and navigate each website, it is not feasible to evaluate all clients of each service. As such, we decided to focus on a subset of their clients by randomly selecting ten clients for each anti-bot service. We ensure that the selected websites do not exhibit any client-side signs (e.g. JavaScript libraries) that would suggest that they are utilizing any anti-bot service, other than the evaluated ones. We also removed websites which, through experimentation, showed signs of additional, server-side software blocking our requests. This is not a challenge since block pages used by anti-bot services are distinctive (Figure 2) and HTML tags, URLs, and variable names within the page source point to the anti-bot company.

1. Test Preparation We target the same attack categories that we discussed in gray box tests. Since we do not have access to the administration panel this time, we devised heuristics to detect being blocked based on the received response.

- **Account Takeover and Credential Brute Force:** We use the same type of scripts that we used in gray box tests. Note that in both experiments we create an account on the target website and only target our own user account during the experiments, for ethical purposes.
- **Content Scraping:** In this attack scenario, we inspect the websites of clients of anti-bot services, and identify content that is a likely target for scraping by malicious crawlers (such as pricing of products and inventory details). We then implement the necessary automation scripts for each website, attempting to scrape 1,000 pages worth of content.

We spent a total of 5 man-months developing automation scripts for all tested websites which could appropriately navigate each website according to our desired tests. Another obstacle that we had to overcome is that, due to the churn of clients of the anti-bot services, we had to often repeat experiments with new randomly sampled websites, as some websites stopped being clients of the services before we were able to finish our experiments.

2. Test Execution Using the same infrastructure, we run our experiments against selected clients of each service. To generalize our results, in addition to Linode, we ran a set of pilot tests from AWS and did not observe significant differences in the results (5% over 60 tests) showing that the choice between popular and less popular cloud providers does not have significant impact on the final results.

3. Post-processing of Results After each test, we inspect our logs and screenshots to locate the number of successful attempts each bot made before getting blocked and to make sure any observed blocking is the outcome of fingerprint-related and behavior-related information that these services gather from our bots and correlate with server-side events.

The extracted information consisting of fingerprint, headers and user actions are used by each anti-bot service to come up with a verdict for each user ID, which their client will use to decide whether they should block the current request. Section 6 discusses the results from this step in more detail. In all cases where we received unexpected responses, we manually inspected them to verify that our scripts were indeed blocked. We define "success" and "failure" as follows:

- **Successful content scraping** is defined as our bot loading the content for 1,000 pages on protected websites containing information that would be worth scraping for attackers.
- **Successful account takeover** is defined as our bot being able to login to a target user account from a different location and fingerprint from the actual user's fingerprint used to register and login to the account. The test is considered to have failed when the tool fails to login with explicit (e.g. "You are blocked") or implicit (e.g. "Incorrect credentials") responses.
- **Successful brute forcing** in our experiment is defined as a bot sending 1,000 login attempts with incorrect credentials and then being able to login with correct credentials. We designed this test in a way that simulates an attacker attempting a large number of incorrect credentials before finding the correct one.

For credential brute-forcing experiments, we distinguish the following cases as being blocked: being blocked with an explicit message, receiving a CAPTCHA in order to login, target user account being locked (note that this is always our account), being rate-limited for a considerable amount of time or not being able to login with correct credentials after brute forcing. The last case is based on the observation that some anti-bot services silently increase the risk score when the noisy brute force behavior is observed, and as a result, prevent the bot from logging in even with correct credentials.

5.3 Ethical Considerations

To understand how real anti-bot services detect malicious bots on their client websites, we cannot avoid sending bot-like traffic to public websites. To conduct these experiments in an ethical fashion we took special care when designing them and conducting them. For content scraping, we access content that is considered public, i.e., it is not behind a registration wall. For account takeovers, we only try to log in to our own account on all websites from a location/fingerprint that is different from the one that we utilized to register that account. Lastly, for account brute forcing, we only make login attempts against our own accounts, never trying to log in into the accounts of other users. We provide ample time between requests (in the order of seconds) allowing our requests to be interleaved with regular traffic received by the evaluated popular websites. Our bots behave as humans and therefore never send any maliciously-crafted input to target web applications.

As a result, we are confident that our experiments did not have any negative consequences, neither for the protected websites, nor for the anti-bot services themselves.

6 Analysis of Results

In this section, we describe the results of our bot experiments on our test websites (gray box testing) and on client websites of popular anti-bot services (black box testing). By combining data across both types of experiments, we uncover shortcomings and flaws of these services. Our focus in this section is on the common patterns across the services that will provide an opportunity for the attackers to bypass their protection. As our study is not meant to promote one product over another, we opt to anonymize the names of the anti-bot services.

6.1 Gray Box Testing Results

From the list of anti-bot services that we initially started with, we were able to obtain accounts from four services: Service #3, Service #4, Service #2 (providing generic anti-bot protection) and Service #1 (providing specialized protection against account fraud). For the first two services, installing their WordPress plugins and including the JavaScript file in all of our pages was sufficient to adopt them. After each client request reaches the webserver, these plugins collect the request context including HTTP headers, cookies and IP addresses and report it back to the anti-bot service through their APIs. Conversely, Service #2 protection is enabled by routing website traffic through their servers by changing DNS records. In the case of Service #1, while the JavaScript code sends back information on each page load, we needed to manually call their API upon authentication and report the event. Upon successful authentication, we then have to query their API to receive a verdict (Allow, Challenge, and Block) that defines what action the service recommends. The reaction to these verdicts is also the website developers' responsibility and can vary for each client website (e.g., showing CAPTCHA, or requesting a second factor of authentication).

Gray Box Content Scraping Results The number of successful content-scraping attempts against the three services which protect against it (Service #3, Service #4, and Service #2) is listed in Table 3. For Service #2, none of our scraping bots was ever blocked, regardless of their location (i.e. Campus vs. Cloud). While further fine-tuning the rate limits might be helpful to block more aggressive bots, as long as bots keep their request number low, they can hide within normal user traffic and remain undetected.

Our results show that Service #3 clearly makes a distinction in its decisions to block bots based on their source IP address. Requests from bots originating from campus IP addresses were strictly more successful, compared to those of Linode datacenters. For Service #4, this observation does not always hold true. While Firefox Stripping (i.e. with fingerprinting script blocked) got worse results when originating from Linode, AppleScript and Mobile scrapers remained undetected. This suggests that Service #3 places more weight on the source IP address in their decision-making model, compared to Service #4. Service #3 trusts IP addresses to the extent that attackers with access to prime IP addresses from outside datacenters (Campus address space, in our case) can scrape content even with trivial tools from the "Basic Bots" category.

Table 3: *Number of successful content scraping attempts (Gray box)*

Service	Tool/IP	Python	PhantomJS	Firefox (Stripping)	Chrome	Chrome (Mouse)	Firefox (Mouse)	Safari	Mobile
Service #2	*Campus*	1000	1000	NA	1000	1000	1000	1000	1000
	Cloud	1000	1000	NA	1000	1000	1000	1000	1000
Service #3	*Campus*	1000	0	1000	1000	1000	1000	1000	1000
	Cloud	4	0	21	23	7	14	24	23
Service #4	*Campus*	21	0	1000	0	1	1	1000	1000
	Cloud	3	1	16	0	0	0	1000	1000

Gray Box Account Fraud Results The account-fraud tests are relevant for all four companies in our gray box experiment. We analyze the results for both account takeover and account brute force tests. The time window of interaction for account takeover is limited to two requests (one to grab the CSRF token and one to login). As a result, features, such as, login history including locations, fingerprint of used devices and fingerprints of bots, are more effective in this scenario compared to behavioral anomaly detection. Table 4 shows the number of successful brute force attempts along with whether the account takeover was successful.

Plain Python outperforms PhantomJS: The first unexpected observation, which is consistent among nearly all tests and anti-bot services, is that PhantomJS has inferior performance to plain Python scripts. For example, all services with client fingerprinting capability block login attempts from PhantomJS. Our hypothesis is that this tool was so much overused by attackers (PhantomJS was the first headless JavaScript-supporting browser) that anti-bot services have enough features to detect it with high confidence. Contrastingly, since Python scripts are not capable of executing JavaScript, anti-bot services give them the benefit of the doubt (e.g. it may be a JavaScript-blocking user) and allow a few requests to go through before taking action.

Safari breaks into all user accounts: Interestingly, none of the services block Safari that is automated by Applescript. To our surprise, the risk score reported by Service #1 for Applescript is very low (17/100) even though a real user never logged into the user account with

an Apple device or Safari. For comparison, this score is in the range of 70-88 for Selenium and 100 (i.e. maximum risk) for PhantomJS.

Table 4: *Number of brute force attempts before getting blocked (Gray box)*
Checkmarks indicate successful account takeover

Service	**Service #1**		**Service #2**		**Service #3**		**Service #4**	
Tool	*Campus*	*Cloud*	*Campus*	*Cloud*	*Campus*	*Cloud*	*Campus*	*Cloud*
Python	1000 ✔	0 ✖	5 ✔	5 ✔	240 ✔	12 ✔	2 ✔	1000 ✔
PhantomJS	0 ✖	0 ✖	5 ✔	5 ✔	0 ✖	0 ✖	0 ✖	0 ✖
Chrome	0 ✖	0 ✖	5 ✔	5 ✔	250 ✔	23 ✔	0 ✖	0 ✖
Firefox	10 ✖	0 ✖	5 ✔	5 ✔	405 ✔	21 ✔	1 ✔	0 ✖
Safari	1000 ✔	0 ✔	5 ✔	5 ✔	334 ✔	13 ✔	1000 ✔	1000 ✔

Not executing JavaScript can be helpful: Finally, using Python, our bots were able to successfully log into user accounts both from Campus and Linode IP addresses with a high degree of success. Service #1 was the only exception which returned the verdict of "Challenge" only for Python requests from Linode.

For brute force tests (Table 4), similar to account takeover results, Service #3 shows higher sensitivity to source IP addresses and blocks requests from Linode more aggressively whereas Service #4 does not penalize Python and Safari-based bots. For Service #1, this transition from Campus to Cloud was enough to mark all our login attempts as malicious and increase their risk scores, shifting them to their next verdict category: from Allow to Challenge and from Challenge to Block.

6.2 Overall Content Scraping Results

In this section, we report on six anti-bot services which were either included in our gray box tests or match our criteria for black box tests, that is: (a) provide overall protection against automated attacks/mention content scraping as a covered use case; (b) have a representative sample of at least 10 client websites. We chose distinct websites that use only the corresponding service out of the known anti-bot solutions. Each website was tested against eight different automation tools among three bot categories and each test included the scraping of 1,000 pages.

Table 5 summarizes our results for content scraping from gray box tests and black box tests. The column named "IP sensitivity" indicates whether using IP addresses from Cloud (Linode) rather than Campus makes the service block our bots earlier. If there is more than 50 % change (i.e., at least half of the websites that did not block us on campus IPs blocked us from cloud IPs), we consider the service to be highly sensitive to cloud IP addresses and if the change is less than 50 %, we say the impact of source IP address is low. Finally if we do not observe a significant difference when moving from Campus to Cloud, we infer that presence/absence of an IP address from a cloud provider, does not have an effect on the blocking decision.

In Tables 5 and 6, partial success is marked with half-filled circles indicating that either some tools within that bot category were not blocked (e.g., PhantomJS was blocked but Python was not), or some websites protected by the same service blocked a tool within a category while others did not. Even though we do not have enough information for a definitive

answer, we opine that the partial difference in behavior among clients of the same anti-bot company is the result of different characteristics of their normal traffic and the dynamic nature of the machine learning models.

Table 5:
Service ability to block content scraping by different bots (Gray box and Black box tests)
(●: Blocked, ◑: Some automation tools can bypass, ○: Failed to block)

Anti-bot Service	Basic bots	Automated browsers	Less Popular FPs	Stripping	IP Sensitivity
Service #2 (G)	○	○	○	NA	None
Service #3 (G)	◑	○	○	NA	High
Service #4 (G)	●	●	○	●	None
Service #5	●	◑	◑	●	None
Service #6	◑	○	○	NA	None
Service #7	◑	◑	○	●	Low

Basic bots: Nearly all services with the exception of Service #2 are able to block "basic bots" to some extent. Among them, Service #5 and Service #4 successfully block both Python and PhantomJS consistently on all their clients.

Automated browsers: Service #4 always detects and blocks automated browsers. While Firefox automated by Selenium goes undetected by Service #5 and Service #7, Selenium Chrome is blocked quickly (in less than 100 requests). Adding page scrolls and mouse moves to our automated browsers only helped with Service #7 which led to scraping 1,000 pages on some clients and getting blocked after around 60 attempts on other clients where we could previously make less than five successful requests.

Less popular fingerprints: Applescript-Safari and Chrome on Android are able to bypass the limitations imposed by most anti-bot services. The only exception is Service #5 where some of their clients block Applescript-Safari while others block Chrome on Android. Even the websites that block either of these tools, do so after 300-600 attempts. Compared to basic bots and automated browsers which made less than 10 successful attempts on Service #5 clients, this demonstrates that unpopular, JavaScript-enabled clients can be significantly more successful in evading detection.

Stripping: Stripping refers to blocking fingerprinting JavaScript when scraping content from the websites. None of the services allowed our bots to scrape more pages when client side fingerprints were stripped, and most of the time we got blocked earlier (e.g., in less than 5 attempts on Service #5). Since Service #2, Service #3 and Service #6 either do not perform client-side fingerprinting via JavaScript or do not block our automated browsers, we can not compare their performance when automated browsers are used and fingerprinting scripts are blocked.

Figure 3 generalizes the performance of different scraping bots across all evaluated services. There we can see that, even though the different traffic patterns of different websites lead to noisy results, there are clear patterns that favor some tools over others. For example, when operating either an Android bot or a Firefox-Selenium one with added mouse moves from "premium" address space, attackers can scrape content from the vast majority of sites and services. For cloud tests, we repeated the experiments for the tools that showed better performance from campus IP addresses.

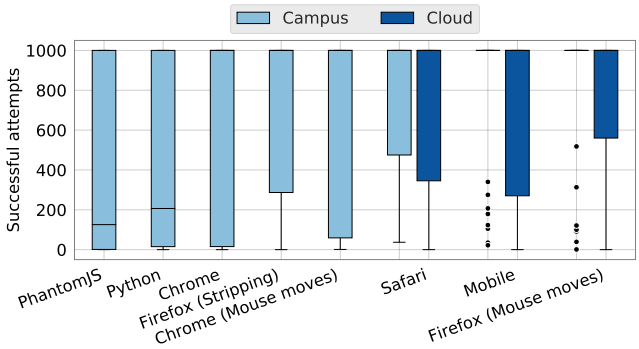

Fig. 3: *Performance of Content Scraping Bots*

6.3 Overall Account Fraud Results

Here, we present our results for the Account Takeover and Brute Force experiments. We analyzed ten anti-bot services which advertise themselves as general anti-bot or account-fraud protection services and for which we could find at least ten distinct client websites that allowed us to register a new user (a requirement for these experiments). We evaluate these services against five tools by performing a total of 2,800 tests.

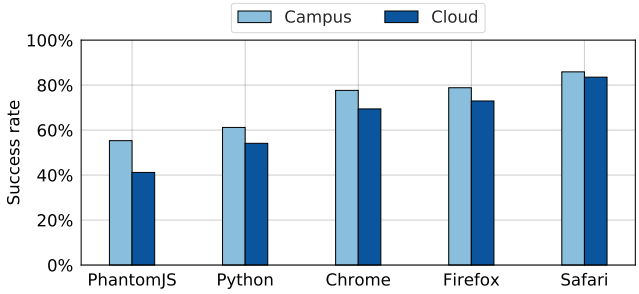

Fig. 4: *Performance of Account Takeover Bots*

Account Takeover Results Overall, 2-8% of websites blocked all our account takeover attempts across all bot categories from Campus and Cloud IP addresses respectively. Applescript-Safari was the most successful tool with 82.5% average success rate. Bots based on Safari automated by AppleScript, were able to break into user accounts with unseen fingerprints (Safari browser) and from new IP addresses. Table 6 summarizes our results for gray box and black box tests for account takeover and brute force tests. The results are sobering. By looking at Figure 4, we observe that because of the absence of a large number of requests to the service during account takeover (attackers have already stolen the credentials and are logging in from a "foreign" environment), most services fail to block the attack. We have already seen that general bot-detection mechanisms fail to block most of the bots right away except basic bots, which also holds true for account takeover attacks. The change in fingerprint and location of the login attempt were not enough to raise an alarm and block the takeover in many of our attempts. As a result, even with the worst-performing bot (i.e. PhantomJS) attackers can successfully conduct an account takeover attack in 40-60% of the time.

Table 6:
Service ability to block Account Takeover by different bots (Gray box and Black box tests)
(●: Blocked, ◑:Some automation tools can bypass, ○: Failed to block)
Brute force: number of websites without any login rate-limiting

Anti-bot Service	Basic bots	Automated browsers	Less Popular FPs	Brute Force	IP Sensitivity
Service #1 (G)	◑	◑	○	6/10	High
Service #2 (G)	○	○	○	0	None
Service #3 (G)	◑	○	○	4/10	High
Service #4 (G)	◑	◑	○	10/10	Low
Service #5	●	●	◑	3/10	High
Service #6	◑	○	○	7/10	Low
Service #7	◑	◑	◑	3/10	Low
Service #8	○	○	○	9/10	None
Service #9	◑	◑	◑	7/10	High
Service #10	●	◑	○	8/10	Low

Credential Brute Force Results The results for this section are summarized in Table 6. Column "Brute Force" in this table refers to the number of websites on which at least one of our bots was able to perform 1,000 brute force attempts against their login forms. This not only shows the lack of defense from anti-bot services but also signifies that neither the website nor the anti-bot service enforce a hard limit on the number of failed attempts (e.g. by account lockout, CAPTCHA or IP address ban). Lu et al. studied the presence of rate limiting mechanisms on top Alexa websites and already pointed out this lack of protection [30]. Our results support Lu et al.'s findings by showing that, even among the websites that actively seek to protect against bot attacks, 30-100% of them do not enforce any type of login rate limiting.

Among all tested services, Service #5 and Service #7 blocked more categories of bots and enforced rate limits on a wider range of tested clients. For Service #4 and Service #8, almost none of their tested clients enforced rate limiting. Interestingly, simple rate-limiting on POST requests to login pages enforced by Service #2, is sufficient to fully prevent brute force attacks, even without any type of client-side fingerprinting.

Contrary to content-scraping results, different tools from automated browsers and less popular fingerprint categories achieve similar results. However, Safari is still the best performing bot. Orthogonally to the type of bot being used, we observe that most anti-bot services become slightly stricter when the bot is sending authentication requests from Cloud IP addresses. For example, the average number of successful requests by Safari, drops from 564 to 433 after transitioning to cloud. This subtle effect is visible in Figure 5. More importantly, the hourglass-like distributions of Figure 5 show that the majority of bruteforce attempts are either blocked in under 400 attempts (the narrow "neck" of the hourglass) or not blocked at all. With the use of more sophisticated tools (right side of the Figure 5), the number of successful 1,000 bruteforce attempts increases.

When combining account takeover and brute force protection, Service #5, Service #7 and Service #9 block bots across all categories. Yet, specific tools are able to evade detection. On the other end, websites using Service #8 in our dataset did not show any specific pattern of blocking bots (except one website that was enforcing a local rate limit to block brute force attempts).

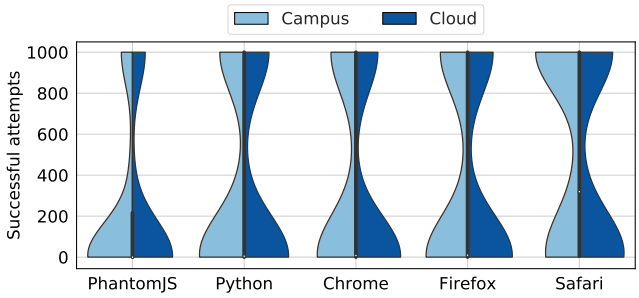

Fig. 5: *Performance of Brute Force Bots*

7 Discussion

In this paper, we conducted the first, large-scale study of commercial anti-bot services that websites can use to detect and protect their content and their users against malicious bots. Using basic bots as well as popular and less popular automated browsers, in conjunction with different types of IP address space (public clouds vs. campus networks), we evaluated — in an ethical fashion — the ability of ten services to detect and block bots on the websites of their customers. While each service has its own specific strengths and weaknesses (as described in Section 5 and Section 6), we can still observe common patterns across services. We describe these patterns (and their implications) below:

Variance across clients of the same service. An unexpected finding of our experiments is that not all clients of the same service block bots in the same way. This could suggest that some services are more sensitive to false positives than others, but it could also suggest misconfigurations from the side of clients of anti-bot services. Our recommendation is that anti-bot services regularly perform bot-based crawling of their own clients and observe whether their own attempts are blocked by their clients. In the cases where blocking is under a configurable threshold, these anti-bot services can reach out to their clients and inquire whether the recorded permissiveness is a conscious choice or merely a misconfiguration.

Browser fingerprinting. Virtually all services rely, to a certain extent, on browser fingerprinting as part of their bot-detection logic. Browser fingerprinting is a powerful mechanism that can be used either constructively (for authentication) or destructively (for unwanted online tracking) to re-identify users (including attackers). Yet it is also susceptible to evasions when attackers are aware of it. When it comes to advanced attackers who can mix and match bots, constructively using browser fingerprinting is more likely to work in a whitelisting fashion (i.e. is the current user's browsing environment, similar to their past browsing environment?) rather than in a blacklisting fashion (i.e. is the current user's fingerprint matching that of a previously-observed, malicious bot?).

PhantomJS is universally recognizable. As we showed in our experiments in Section 5, PhantomJS is universally recognizable by anti-bot services and often performs worse than simple bots that do not support JavaScript at all. Even though this is desirable for detecting attackers abusing PhantomJS, academic researchers have also extensively relied on PhantomJS for web-security [37, 42, 45, 48, 51] and web-privacy [13, 43] studies. Assuming the increasing adoption of anti-bot services, this means that the results obtained through PhantomJS-related crawling will be decreasingly accurate. In the short term, we recommend that researchers

avoid using PhantomJS in favor of newer and more complete crawling tools, such as, headless Chrome and OpenWPM [21]. In the long term, we need both the technical means to evaluate the fingerprintability of crawling frameworks used for research, as well as a discussion between stake-holders on how crawling-based studies should be best conducted.

8 Responsible Disclosure

During this study we observed behaviors that can either be attributed to customer-website misconfigurations of an anti-bot service (i.e. customers do not fully take advantage of the detection capabilities of anti-bot services) or can be blind spots within the detection models of the evaluated services. As such, we contacted 7 services with what we regarded as high-impact misconfigurations or security issues on their client websites in December 2019. During these communications, our goal was to share our findings and obtain more information about the design decisions and details that we could not observe as outsiders. Ultimately, three services (ThreatMetrix, DataDome, and Castle) reached back to us. We have shared the list of vulnerable target websites in our study and our bot scripts with the anti-bot services upon their request and we are in continuous conversation with them. We hope that this information will be used to increase the accuracy and coverage of these services.

9 Related Work

Research-wise, despite its potential for abuse, bot identification has only attracted limited research which, given the adaptations from bot authors, can quickly become dated. Existing attempts to differentiate crawlers from real users rely on differences in their navigational patterns, the percentage of HTTP methods in requests, the types of links requested, and the timing between requests [24, 29, 46]. These features are then used in supervised machine-learning algorithms trained using ground truth that the authors of each paper were able to procure, typically by manually labeling traffic of one or more webservers to which they had access. Xie et al. propose an offline method for identifying malicious crawlers by searching for clusters of requests towards non-existent resources [52]. Next to ML-based methods, Park et al. [39] investigated the possibility of detecting malicious web crawlers by looking for mouse movement, the loading of Cascading Style Sheets, and the following of an invisible link that is present in the HTML code of a page yet is invisible to regular users. McKenna [31] recently experimented with more types of invisible links and resources but was unable to gauge their effectiveness due to size and duration limitations of their study.

Interestingly, the majority of work on bot detection predates browser fingerprinting despite the latter appearing as early as 2010 [12, 13, 18, 20, 21, 23, 27, 36] even though as we showed throughout this paper, *all but one* of the evaluated anti-bot services heavily rely on fingerprinting for identifying bots.

All the prior research that focused on adding new attributes to a fingerprint [22, 33, 35, 44], notably techniques like canvas [34], AudioContext [21] or WebGL [16] fingerprinting, is especially relevant to anti-bot services as it could offer more ways to distinguish a bot from a regular user. Moreover, there exists machine-learning approaches to link fingerprint evolutions over time [49] which could be used to track changes in a bot fingerprint. Relying on fingerprinting techniques, Bursztein et al. proposed Picasso, a tool aimed at identifying inorganic traffic through canvas fingerprinting [14].

Jueckstock et al. introduced VisibleV8, an instrumented Chromium based browser that is capable of monitoring dynamic JavaScript API calls [25]. The authors found that 29% of

top 50k Alexa websites probe for artifacts of automated browsing environment but do not evaluate the usage of these artifacts and whether these websites can detect different types of bots in practice. Vastel et al. study the presence of bot-detection artifacts over the Alexa top 10K websites [50]. While they focus on fingerprinting behavior of anti-bot systems, our study systematically evaluates the overall benefits and drawbacks of existing anti-bot approaches as they are deployed in the wild. Moreover, we model real world attack scenarios whereas previous work focused on the fingerprinting surface of browsers and blocking behavior when visiting target websites.

10 Conclusion

In this paper, we reported on the first analysis of anti-bot services for the web. By isolating and analyzing the JavaScript code which the clients of anti-bot services need to utilize, we identified near universal-reliance on browser fingerprinting, including recently-proposed fingerprinting techniques, as well as checks for the consistency of the presented fingerprints. Through large-scale, black box and gray box analyses of each service using off-the-shelf automation tools as well as less-popular automated environments, we quantified the ability of these services to detect and block bots. We discovered that many services perform poorly and browsers that are less commonly automated (i.e Safari on Mac and Chrome on Android) can achieve an overall success rate of 80% during content-scraping attempts. We also discovered that the location of a bot on a public cloud is secondary to its fingerprint and only 4 services are sensitive to the location of source IP address. This allows attackers to launch massive bot campaigns by renting low-cost virtual machines on public data centers.

Overall, our findings suggest that existing services can stop basic bots, but are currently not capable of blocking specialized tools and even the less popular automated browsers, which can bypass the protection of around 75% of content-scraping targets. As such, they cannot substantially limit determined attackers. At the same time, our findings are relevant for all research involving the crawling of websites since websites that utilize anti-bot services may be able to identify the tools used by researchers (such as PhantomJS) and thereby evade accurate analysis.

Acknowledgements: We thank the anonymous reviewers for their helpful feedback. This work was partially supported by a gift from Amazon and the National Science Foundation (NSF) under grants CNS-1813974, CNS-1617902, and CMMI-1842020.

References

1. Panopticlick's fonts2.swf, https://github.com/EFForg/panopticlick-python
2. Nightmare (2014), https://github.com/segmentio/nightmare
3. Evercookie (2017), https://github.com/samyk/evercookie
4. Figerprint2js (2017), https://github.com/Valve/fingerprintjs2
5. Modernizr (2017), https://modernizr.com/
6. Nerdydata: Search engine for source code (2017), https://nerdydata.com
7. PluginDetect (2017), http://www.pinlady.net/PluginDetect/
8. Publicwww: Search engine for source code (2017), https://publicwww.com
9. Uber's Massive Scraping Program Collected Data About Competitors Around The World (2017), https://gizmodo.com/ubers-massive-scraping-program-collected-data-1820887947
10. BuiltWith Technology Lookup (2018), https://builtwith.com/
11. How Bots Are Disrupting Airline Ticket Sales (2019), https://www.eweek.com/enterprise-apps/how-bots-are-disrupting-airline-ticket-sales

12. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In: Proceedings of the 2014 ACM CCS Conference
13. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., Preneel, B.: FPDetective: dusting the web for fingerprinters. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 1129–1140. CCS '13, ACM, New York, NY, USA (2013)
14. Bursztein, E., Malyshev, A., Pietraszek, T., Thomas, K.: Picasso: Lightweight device class fingerprinting for web clients. In: Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices (2016)
15. Canali, D., Balzarotti, D.: Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web. In: 20th Annual Network & Distributed System Security Symposium (NDSS 2013) (2013), https://hal.archives-ouvertes.fr/hal-00799082
16. Cao, Y., Li, S., Wijmans, E.: (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In: 24nd Annual Network and Distributed System Security Symposium, NDSS (2017)
17. Cloudflare: The web performance & security company. https://www.cloudflare.com
18. Das, A., Acar, G., Borisov, N., Pradeep, A.: The web's sixth sense: A study of scripts accessing smartphone sensors. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1515–1532. ACM (2018)
19. DistilNetworks: Bad Bot Report (2018), https://resources.distilnetworks.com/travel/2018-bad-bot-report
20. Eckersley, P.: How Unique is Your Web Browser? In: Proceedings of the 10th International Conference on Privacy Enhancing Technologies. PETS'10, Springer-Verlag, Berlin, Heidelberg (2010)
21. Englehardt, S., Narayanan, A.: Online Tracking: A 1-million-site Measurement and Analysis. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1388–1401. CCS '16, ACM, New York, NY, USA (2016)
22. Fifield, D., Egelman, S.: Fingerprinting web users through font metrics. In: Proceedings of the 19th international conference on Financial Cryptography and Data Security (2015)
23. Gómez-Boix, A., Laperdrix, P., Baudry, B.: Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In: WWW 2018: The 2018 Web Conference (Apr.)
24. Jacob, G., Kirda, E., Kruegel, C., Vigna, G.: Pubcrawl: Protecting users and businesses from crawlers. In: USENIX Security Symposium. pp. 507–522 (2012)
25. Jueckstock, J., Kapravelos, A.: VisibleV8: In-browser monitoring of JavaScript in the wild. Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC pp. 393–405 (2019)
26. Kelley, P.G., Komanduri, S., Mazurek, M.L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L.F., López, J.: Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. Proceedings - IEEE Symposium on Security and Privacy (2012)
27. Laperdrix, P., Rudametkin, W., Baudry, B.: Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. In: 37th IEEE Symposium on Security and Privacy (S&P 2016)
28. Lerner, A., Simpson, A.K., Kohno, T., Roesner, F.: Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In: USENIX Security 2016
29. Lourenço, A.G., Belo, O.O.: Catching web crawlers in the act. In: Proceedings of the 6th international Conference on Web Engineering. pp. 265–272. ACM (2006)
30. Lu, B., Zhang, X., Ling, Z., Zhang, Y., Lin, Z.: A measurement study of authentication rate-limiting mechanisms of modern websites. In: Proceedings of the 34th Annual Computer Security Applications Conference. ACSAC '18 (2018)
31. McKenna, S.F.: Detection and classification of Web robots with honeypots. Ph.D. thesis, Monterey, California: Naval Postgraduate School (2016)
32. Melicher, W., Ur, B., Segreti, S.M., Komanduri, S., Bauer, L., Christin, N., Cranor, L.F.: Fast, lean, and accurate: Modeling password guessability using neural networks. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 175–191 (2016)
33. Mowery, K., Bogenreif, D., Yilek, S., Shacham, H.: Fingerprinting Information in JavaScript Implementations. In: Wang, H. (ed.) Proceedings of W2SP 2011. IEEE Computer Society (May 2011)

34. Mowery, K., Shacham, H.: Pixel Perfect: Fingerprinting Canvas in HTML5. In: Proceedings of W2SP 2012 (2012)
35. Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., Wien, F.C.: Fast and reliable browser identification with javascript engine fingerprinting. In: Web 2.0 Workshop on Security and Privacy (W2SP). vol. 5 (2013)
36. Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., Vigna, G.: Cookieless Monster: Exploring the Ecosystem of Web-Based Device Fingerprinting. In: Proceedings of the 2013 IEEE Symposium on Security and Privacy. pp. 541–555 (2013)
37. Nikiforakis, N., Maggi, F., Stringhini, G., Rafique, M.Z., Joosen, W., Kruegel, C., Piessens, F., Vigna, G., Zanero, S.: Stranger danger: Exploring the ecosystem of ad-based url shortening services
38. Olejnik, Ł., Acar, G., Castelluccia, C., Diaz, C.: The leaking battery. In: Data Privacy Management, and Security Assurance (2015)
39. Park, K., Pai, V.S., Lee, K.W., Calo, S.B.: Securing web service by automatic robot detection. In: USENIX Annual Technical Conference, General Track. pp. 255–260 (2006)
40. PhantomJS - Scriptable Headless Browser. `http://phantomjs.org/`
41. Shekyan, S.: Detecting PhantomJS Based Visitors. `https://blog.shapesecurity.com/2015/01/22/detecting-phantomjs-based-visitors/` (2015)
42. Srinivasan, B., Kountouras, A., Miramirkhani, N., Alam, M., Nikiforakis, N., Antonakakis, M., Ahamad, M.: Exposing search and advertisement abuse tactics and infrastructure of technical support scammers. In: Proceedings of the 2018 World Wide Web Conference
43. Starov, O., Gill, P., Nikiforakis, N.: Are you sure you want to contact us? quantifying the leakage of pii via website contact forms. Proceedings on Privacy Enhancing Technologies (2016)
44. Starov, O., Nikiforakis, N.: XHOUND: Quantifying the Fingerprintability of Browser Extensions. In: 38th IEEE Symposium on Security and Privacy (S&P 2017). San Jose, United States (2017)
45. Tajalizadehkhoob, S., Van Goethem, T., Korczyński, M., Noroozian, A., Böhme, R., Moore, T., Joosen, W., van Eeten, M.: Herding vulnerable cats: a statistical approach to disentangle joint responsibility for web security in shared hosting. In: Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security. ACM (2017)
46. Tan, P.N., Kumar, V.: Discovery of web robot sessions based on their navigational patterns. In: Intelligent Technologies for Information Analysis (2004)
47. Thomas, K., McCoy, D., Grier, C., Kolcz, A., Paxson, V.: Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In: USENIX Security (2013)
48. Van Goethem, T., Piessens, F., Joosen, W., Nikiforakis, N.: Clubbing seals: Exploring the ecosystem of third-party security seals. In: Proceedings of the 2014 ACM CCS Conference
49. Vastel, A., Laperdrix, P., Rudametkin, W., Rouvoy, R.: FP-STALKER: Tracking Browser Fingerprint Evolutions. In: 39th IEEE Symposium on Security and Privacy (S&P 2018)
50. Vastel, A., Rudametkin, W., Rouvoy, R., Blanc, X.: FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In: Starov, O., Kapravelos, A., Nikiforakis, N. (eds.) NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb'20) (2020)
51. Vissers, T., Van Goethem, T., Joosen, W., Nikiforakis, N.: Maneuvering around clouds: Bypassing cloud-based security providers. In: Proceedings of the 22nd ACM CCS Conference (2015)
52. Xie, G., Hang, H., Faloutsos, M.: Scanner hunter: Understanding http scanning traffic. In: Proceedings of the 9th ACM symposium on Information, computer and communications security (2014)