Fingerprint surface-based detection of web bot detectors*

Hugo Jonker[1,3], Benjamin Krumnow[2,1], and Gabry Vlot[1]

[1] Open Universiteit, Heerlen, Netherlands
[2] Technische Hochschule Köln, Germany
[3] iCIS Institute, Radboud University, Nijmegen, Netherlands

Abstract. Web bots are used to automate client interactions with websites, which facilitates large-scale web measurements. However, websites may employ web bot detection. When they do, their response to a bot may differ from responses to regular browsers. The discrimination can result in deviating content, restriction of resources or even the exclusion of a bot from a website. This places strict restrictions upon studies: the more bot detection takes place, the more results must be manually verified to confirm the bot's findings.
To investigate the extent to which bot detection occurs, we reverse-analysed commercial bot detection. We found that in part, bot detection relies on the values of browser properties and the presence of certain objects in the browser's DOM model. This part strongly resembles browser fingerprinting. We leveraged this for a generic approach to detect web bot detection: we identify what part of the browser fingerprint of a web bot uniquely identifies it as a web bot by contrasting its fingerprint with those of regular browsers. This leads to the *fingerprint surface* of a web bot. Any website accessing the fingerprint surface is then accessing a part unique to bots, and thus engaging in bot detection.
We provide a characterisation of the fingerprint surface of 14 web bots. We show that the vast majority of these frameworks are uniquely identifiable through well-known fingerprinting techniques. We design a scanner to detect web bot detection based on the reverse analysis, augmented with the found fingerprint surfaces. In a scan of the Alexa Top 1 Million, we find that 12.8% of websites show indications of web bot detection.

1 Introduction

There exist various tools to visit websites automatically. These tools, generically termed web bots, may be used for benign purposes, such as search engine indexing or research into the prevalence of malware. They may also be used for more nefarious purposes, such as comment spam, stealing content, or ad fraud. Benign websites may wish to protect themselves from such nefarious dealings, while malicious websites (e.g., search engine spammers) may want to avoid detection. To that end, both will deploy a variety of measures to deter web bots.

* Authors in alphabetic order

There is a wide variety of measures to counter bots, from simple countermeasures such as rate limiting to complex, such as behavioural detection (mouse movements, typing rates, etc.). The more different a web bot is, the simpler the measures needed to detect it. However, modern web bots such as Selenium allow a scraper to automate the use of a regular browser. Such a web bot thus more closely resembles a regular browser. To determine whether the visitor is a web bot may still be possible, but requires more information about the client side. Interestingly, more advanced countermeasures allow a website to respond more subtly. Where rate limiting will typically block a visitor, a more advanced countermeasure may (for example) omit certain elements from the returned page.

A downside of detection routines is that they affect benign web bots as much as malicious web bots. Thus, it is not clear whether a web bot 'sees' the same website as a normal user would. In fact, it is known that automated browsing may result in differences from regular browsing (e.g. [WD05, WSV11, ITK+16]). Currently, the extent of this effect is not known. Nevertheless, most studies employing web bots assume that their results reflect what a regular browser would encounter. Thus, the validity of such studies is suspect.

In this paper, we investigate the extent to which such studies may be affected. A website can only tailor its pages to a web bot, if it detects that the visitor is indeed a web bot. Therefore, studies should treat websites employing web bot detection differently from sites without bot detection. This raises the question of how to detect web bots. We have not encountered any studies focusing exclusively on detecting whether a site uses web bot detection.

Contributions. In this paper, we devise a generic approach to detecting web bot detection, which leads to the following 4 main contributions. (1) First, we reverse analyse a commercial client-side web bot detector. From this, we observe that specific elements of a web bot's browser fingerprint are already sufficient to lead to a positive conclusion in this particular script. This, in turn, suggests that the browser fingerprint of web bots is distinguishable from the browser fingerprint of regular browsers – and that (some of) these differences are used to detect web bots. We create a setup to capture all common such differences in browser fingerprint. We call this collection of fingerprint elements that distinguish a web bot from a regular browser the *fingerprint surface*, analogous to Torres et al. [TJM15]. We use our setup to (2) determine the fingerprint setup of 14 popular web bots. Using those fingerprint surfaces as well as best practices, we (3) design a bot-detection scanner and scan the Alexa Top 1 million for fingerprint-based web bot detection. To the best of our knowledge, we are the first to assess the prevalence of bot detection in the wild. Finally, we (4) provide a qualitative investigation of whether websites tailor content to web bots.

Availability. The results of the web bot fingerprint surfaces and the source code used for our measuring the prevalence of web bot detectors are publicly available for download from http://www.gm.fh-koeln.de/~krumnow/fp_bot/index.html.

2 Related work

Our work builds forth on results from three distinct fields of research: browser fingerprinting, web bot detection techniques and *website cloaking* – the practice where a website shows different content to different browsers.

Browser fingerprinting. The field of browser fingerprinting evolved from Eckersley's study into using browser properties to re-identify a browser [Eck10]. He was able to reliably identify browsers using a only few browser properties. Since then, others have investigated which further properties and behaviour may be leveraged for re-identification. These include JavaScript engine speed [MBYS11], detecting many more fonts [BFGI11], canvas fingerprinting [MS12], etc. Nikiforakis et al. [NKJ+13] investigated the extent to which these were used in practice, referring to a user's fingerprintable surface without making this notion explicit. Later, Torres et al. [TJM15] updated and expanded the findings of Nikiforakis et al., and introduced the notion of *fingerprint surface* as those elements of a browser's properties and behaviour that distinguish it from other browsers. In this paper, we leverage this notion to devise a generic detection mechanism for web bot detection.

Web bot detection techniques. Many papers have suggested solutions to detect web bots, including [CGK+13, SD09, BLRP10], or to prevent web bots from interacting with websites, e.g. [vABHL03, VYG13]. While these works achieved satisfying results within their experimental boundaries, it is unclear [DG11] which approaches work sufficiently well in practice. Bot detection approaches typically focus on behavioural differences between humans and web bots. For example, Park et al. [PPLC06] use JavaScript to detect missing mouse and keyboard events. On a different level, Xu et al. [XLC+18] contrasted the traffic generated by web bots with that generated by regular browsers. In contrast, the detection methods investigated in this work focus on technical properties, such as browser attributes, that differ between regular browsers and web bot(-driven) browsers.

Website cloaking. Website cloaking is the behaviour of websites to deliver different content to web bots than to regular browsers [GG05b]. Cloaking has mostly been studied in relation to particular cloaking objectives. For example, Wu and Davison [WD05] investigated cloaking in the context of search engine manipulation by crawling sites with user agents for regular browsers and web bots. They estimate that 3% in their first (47K) and 9% in their second set of websites (250K) engage in this type of cloaking. Invernizzi et al. [ITK+16] analysed server-side cloaking tools to determine their capabilities. Based on this, they created a scraping method to circumvent the identified cloaking techniques. Within 136K sites, they found that 11.7% of these URLs linked to cloaked sites. Pham et al. [PSF16] used a similar method as Wu and Davison, but focused on user agent strings. By alternating user agents of a web crawler, they found that user agent strings referring to web bots resulted in significantly more (about 4×) HTTP error codes than user agent strings of regular browsers. Interestingly, user agent strings of a relatively unknown web bot framework worked better than using an empty string. They even outperformed strings of a regular browser.

Fingerprint-detection scanners. Two studies created scanners to detect fingerprinting in the wild. Acar et al. created FPDetective [AJN+13], a framework for identifying fingerprinters in the wild. FPDetective relies on a modified version of the WebKit rendering engine to log access to a handful of DOM properties, plugins and JavaScript methods. A downside to modifying the rendering engine was that browsers regularly update their rendering engine. Acar et al. encountered this, as the Chromium project already moved to another rendering engine during their study. Englehardt and Narayan later developed the OpenWPM framework [EN16] for measuring privacy-related aspects on websites. Their framework is based on Selenium, a program which can automate interactions with and gather data from a variety of browsers. Compared to FPDetective, this allows for a more generic approach (e.g., using multiple browsers), as well as making use of the same browser a user would. For those reasons, our scanner is build on top of OpenWPM.

3 Reverse analysis of a commercial web bot detector

In our search for sites that engage in web bot detection, we encountered a site that allegedly[4] can detect and block Selenium-based visitors. We verified that this site indeed blocks Selenium-based visitors by visiting the site with user- and Selenium-Chromedriver-driven browsers systematically. We investigated JavaScript files used on this site and analysed the page's traffic. The traffic analysis showed that several communications back to the host contained references to 'distil', e.g. in file names (`distil_r_captcha_util.js`) or in headers `X-Distil-Ajax: ...`). This was due to two of the scripts originating from Distil Networks, a company specialised in web bot detection, and thus the likely cause of the observed behaviour. We manually de-obfuscated these scripts by using a code beautifier and translating hex-encoded strings, after which we could follow paths through the code. This allowed us to identify a script that provided the following three main functionalities:

– **Behaviour-based web bot detection.** We found multiple event handlers added to JavaScript interaction events. These cover mobile and desktop specific actions, such as clicks, mouse movements, a device's orientation, motion, keyboard and touch events.

– **Code injection routines.** The traffic analysis revealed frequent communication with the first party server. Within this traffic we found fingerprint information and results of web bot detection. This would allow a server to carry out additional server-side bot detection. We further identified routines, that enable the server to inject code in response to a positive identification. In our test, this resulted in a CAPTCHA being included on the page.

– **DOM properties-based web bot detection.** Lastly, we found that multiple built-in objects and functions are accessed via JavaScript (e.g., see Listing 1). Some of the properties accessed thusly are commonly used by fingerprint-

[4] https://stackoverflow.com/questions/33225947/can-a-website-detect-when-you-are-using-selenium-withchromedriver

```
// Example of original, obfuscated code:
// array containing literals used throughout the source code
var _ac = ["\x72\x75\x6e\x46\x6f\x6e\x74\x73",
    "\x70\x69\x78\x65\x6c\x44\x65\x70\x74\x68", .....]

// Example of de-obfuscation
// obfuscated: window[_ac[327]][_ac[635]][_ac[35]](_ac[436])
// de-obfuscated:
window[document]["documentElement"]["getAttribute"]("selenium")

// Example of de-obfuscated and beautified code
sed: function() {
  var t;
  t = window["$cdc_asdjflasutopfhvcZLmcfl_"] || \
    document["$cdc_asdjflasutopfhvcZLmcfl_"] ? "1" : "0";

  var e;
  e = null != window["document"]["documentElement"]\
    ["getAttribute"]("webdriver") ? "1" : "0";
  ...
}
```

Listing 1. Examples from bot-detection script.

ers [TJM15]. We also found code to determine the existence of specific bot-only properties, such as the property `document.$cdc_asdjflasutopfhvcZLmcfl_` (a property specific to the ChromeDriver). Keys from the `window` and `document` objects were acquired by Distil. Moreover, a list of all supported mime types was also collected (via `navigator.MimeTypes`).

Moreover, we investigated whether changing the name of this specific property affects bot detection. We modified ChromeDriver to change this property's name and used the modified driver to access the site in question 30 times. With the regular ChromeDriver, we always received "bot detected" warnings from the second visit onwards. With the modified ChromeDriver, we remained undetected.

4 A generic approach to detecting web bot detection

From the reverse analysis, we learned that part of Distil's bot detection is based on checking the visitor's browser for properties. Some of these properties are commonly used in fingerprinting, others are unique to bots. Moreover, in testing with a modified ChromeDriver, we found that the detection routines were successfully deceived by only changing one property. This implied that at least some detection routines used by Distil fully rely on specifics of the browser fingerprint. Moreover, both FPDetective [AJN+13] and OpenWPM [EN16] checked whether a website accesses specific browser properties. By combining these findings, we develop an approach to detecting Distil-alike bot-detection on websites.

To turn this into a more generic approach that will also detect unknown scripts, we expand what properties we will scan for. The properties that are used to detect a web bot will vary from one web bot to another. To detect web bot detection for a specific web bot, we first determine its fingerprint surface and then incorporate those properties that are in its fingerprint surface into a

dedicated scanner. Remark that properties and variables that are unique to the fingerprint of a specific web bot serve no purpose on a website, unless the website is visited by that specific web bot and the site aims to change its behaviour when that occurs. Therefore, we hold that if a portion of a fingerprint is unique to a web bot, any site that checks for or operates on that portion is trying to detect that web bot.

With that in mind, we designed and developed a scanner based on the discovered fingerprint surfaces. This scanner thus allows us to scan an unknown site and determine if it is using fingerprint-based web bot detection.

Note that this design does not incorporate stealth features to hide its (web bot) nature from visited sites. To the best of our knowledge, this is the first study to investigate the scale of client-side web bot detection. As such, we expect web bot detection to focus on other effects than hiding its presence. Therefore, we nevertheless deemed this approach sufficient for a first approximation on the scope of client-side web bot detection.

5 Fingerprint surface of web bots

A fingerprint surface is that part of a browser fingerprint which distinguishes a specific browser or web bot from any other. A naive approach to determining a fingerprint surface is then to test a gathered browser fingerprint against all other fingerprints. However, layout engine and JavaScript engine tend to be reused by browsers. The fingerprints of browsers that use the same engines will have large overlap. Thus, to determine the fingerprint surface, it suffices to only explore the differences compared to browsers with the same engines. For example: the property `document.$cdc_asdjflasutopfhvcZLmcfl_` is present in Chrome and Chromium only when used via ChromeDriver, otherwise not.

Thus, we classify browsers and web bots into browser families, according to the used engines. We then gathered the fingerprint of a bot-driven browser, and compared it with regular browsers from the same family. Only properties that are unique to the web bot in this comparison are part of its fingerprint surface. Interestingly, we found that every browser that uses the same rendering engine, also uses the same JavaScript engine. Thus, for the examined browsers, no fingerprint differences can arise from differences in JavaScript engine.

We set up a fingerprinting website to collect the various fingerprints. For fingerprinting, we extended fingerprint2.js[5], a well-known open source browser fingerprinting package, as discussed below. We visited this site with a wide variety of user- and bot-driven browsers, and so were able to determine the fingerprint surfaces of 14 web bots.

[5] https://github.com/Valve/fingerprintjs2

Browser family	*Regular browser*	*Automated browser*
Webkit	Safari	PhantomJS,
	Chrome (v1–v26)	Sel. + WebDriver
Blink	Chrome (v27+),	Puppeteer + Chrome,
	Chromium	NightmareJS,
	Opera (v15+)	Sel. IDE,
	Edge (from mid-2019)	Sel. + WebDriver
Gecko	Firefox	Sel. IDE,
		Sel. + WebDriver
Trident	Internet Explorer	Sel. + WebDriver
EdgeHTML	Edge (till mid-2019)	Sel. + WebDriver

Table 1. Classification of browsers based upon rendering engine.

5.1 Determining the browser family of web bots

In our classification, we omitted bot frameworks that do not use complete rendering engines [GG05a] to build the DOM tree. We included frameworks popular amongst developers and/or in literature, specifically:

- PhantomJS: a headless browser based on WebKit, the layout engine of Safari. PhantomJS is included as it is used in multiple academic studies, even though its development is currently suspended[6].
- NightmareJS: An high-level browser automation library using Electron[7] as a browser. It allows to be run in headless mode or with a graphical interface.
- Selenium WebDriver: a tool to automate browsers. There are specific drivers for each of the major browsers.
- Selenium IDE: Selenium available as plugin for Firefox and Chrome.
- Puppeteer: A Node library to control Chrome and Chromium browsers via the DevTools Protocol. DevTools Protocol allows to instrument Blink-based browsers.

This leads to the classification shown in Table 1. Browsers from different browser families use different rendering and JavaScript engines, which will lead to differences in their browser fingerprints. However, all browsers within one browser family use the same rendering and JavaScript engines. This means their browser fingerprints are comparable: differences in these fingerprints can only originate from the browsers themselves, not from the underlying engines.

5.2 Determining the fingerprint surface

We use the above classification of browser families to determine the fingerprint surface of the listed web bots. To determine the complete fingerprint surface

[6] https://github.com/ariya/phantomjs/issues/15344

[7] Electron is a framework for making stand-alone apps using web technologies. It relies on Chromium and Node.js.

– Plugin Enumeration	– CPU	– DNT User Choice
– Font Detection	– OpenDatabase	– Flash Enabled
– User-Agent	– Canvas fingerprint	– colorDepthKey
– HTTP Header	– Mime-type Enumeration	– HTML body behaviour
– ActiveX + CLSIDs	– IndexedDB	– Physical px ratio to CSS px
– Device Memory (RAM)	– WebGL Fingerprinting	– **Window object keys**
– Screen resolution	– AdBlock	– **Document object keys**
– Available screen resolution	– Language inconsistencies	– **Navigator properties**
– Browser Language	– OS inconsistencies	– ***StackTrace***
– DOM Storage	– Browser inconsistencies	– ***Missing image properties***
– IP address	– Resolution inconsistencies	– ***Sandboxed*** `XMLHttpRequest`
– Navigator platform	– Timezone	– ***Autoclosing dialogs***
– Touch support	– Number of cores	– ***Availability of bind engine***

Table 2. Browser fingerprint gathered. Newly added properties are marked in bold. Bold italic elements resulted from discussions on best practices. A full explanation can be found in Appendix B.1.

is infeasible, as already noted by Nikiforakis et al. [NKJ+13] and Torres et al. [TJM15]. To wit: a fingerprint is a form of side channel for re-identification. As it is infeasible to account for all unknown side channels, it is not feasible to establish a complete fingerprint surface. Hence we follow a pragmatic approach to identifying the fingerprint surface (much like the aforementioned studies). We use an existing fingerprint library and extend[8] it to account for the additional fingerprint-alike capabilities encountered in the analysis of the commercial bot detector, listed below, as well as best practices for bot detection encountered online. The fingerprint surface collected by the tool is shown in Table 2. The updates added due to the reverse analysis are:

- All keys from the `window` and `document` objects.
- A list of all mimetypes supported by the browser.
- A list of all plugins supported by the browser.
- All keys and values of the `navigator` object.

The test site hosting this fingerprint script was then visited with each browser and web bot from the browser family. Only properties that differed between members of the same browser family constitute elements of those browsers' fingerprint surfaces.

Remark that not all deviations in fingerprint lead to a fingerprintable surface. For example, an automated browser may offer a different resolution from a regular browser, which is nevertheless a standard resolution (e.g., 640x480). We thus manually evaluated the resulting deviations between the fingerprints of one browser family and, for each web bot, determined its fingerprint surface accordingly.

[8] https://github.com/bkrumnow/BrowserBasedBotFP/blob/master/public/js/fingerprint.js

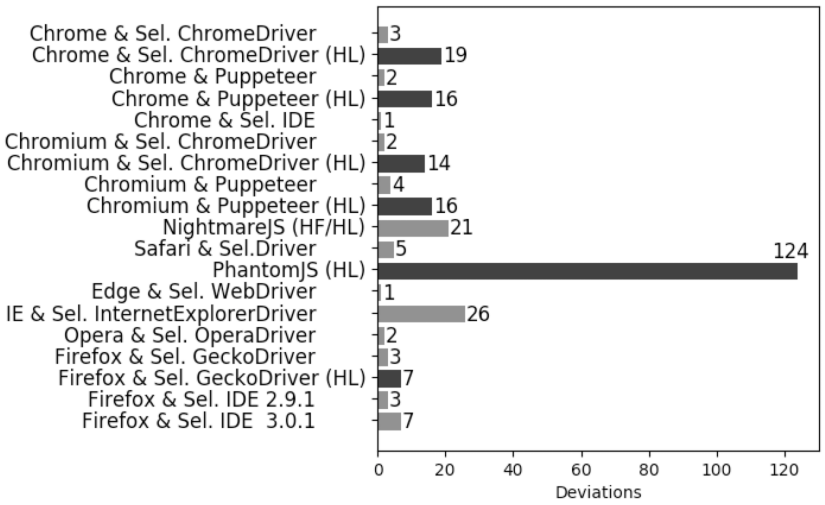

Fig. 1. Browser-wise comparison of the number of deviations. Bars for headless browsers are depicted in black.

5.3 Resulting fingerprint surfaces

Several web bots support *headless* (HL) mode. This mode functions similarly to normal operation of the web bot, but does not output its results to a screen. In total, we determined the fingerprint surfaces of 14 web bots (full fingerprint surfaces available online[9]). Together with variants due to HL mode, this resulted in 19 fingerprint surfaces. We found both newly introduced properties and existing properties where the bot-browser has distinctive values. Figure 1 depicts the number of deviations (i.e., the number of features in the identified fingerprint surface) of the tested web bots. As can be seen, PhantomJS has many deviations. Another finding is that headless mode leads to a greater number of deviations. This happens for all web bots except for NighmareJS.

The results of our fingerprint gathering differed on several points from the results of the reverse analysis. Specifically: for several of the tests used by Distil, we did not encounter any related fingerprint. We investigated this discrepancy by conducting source code reviews of web bot frameworks to trace such tests back to specific web bot frameworks. We found several properties[10] that were no longer in the versions of the frameworks we tested, but were present in other versions (older versions or derived versions such as Selendroid). This underscores the incompleteness of the derived fingerprint surface: updates to web bot frameworks will thus result in changes to the fingerprint surface.

[9] http://www.gm.fh-koeln.de/~krumnow/fp_bot/fp_deviations.html

[10] *webdriver_evaluate*, *webdriver-evaluate*, *fxdriver_unwrapped*, *$wdc*, *domAutomation* and *domAutomationController*

Table 3 shows an example of a set of deviations and the resulting fingerprint. It lists deviations found by comparing Chrome with a headless Selenium-Webdriver-driven Chrome browser. The deviations listed under the UserAgent string (equally to request headers), window and document keys are unique properties and values, that together build the fingerprint surface. Other properties, such as missing plugins or screen resolutions, might be useful indicators for a detector, but are not unique for web bots.

Property	*Chrome*	*Sel. WebDriver Chrome (HL)*
UserAgent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) **Chrome**/ 64.0.3282.140 Safari/537.36	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) **HeadlessChrome**/ 64.0.3282.140 Safari/537.36
Resolution	1920x975	640x480
Available Resolution	1855x951	640x480
Language inconsistencies	False	True
Canvas fp	-	Deviates
Plugins	Chrome PDF Plugin::Portable Document Format::application/x-google-chrome-pdf pdf,...	None
MIME types	{"0":{},"1":{},"2":{},"3":{},...}	{}
Request headers	"content-length":"**65515**", "user-agent":"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/ 537.36 (KHTML, like Gecko) **Chrome**/64.0.3282.140 Safari/537.36", **"accept-language":"en-US,en;q=0.9"**	"content-length":"**64980**", "user-agent":"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/ 537.36 (KHTML, like Gecko) **HeadlessChrome**/64.0.3282.140 Safari/537.36"
Window keys	-	Missing: **chrome**, **attr**
Document keys	-	Added: **$cdc_asdjflasutopfhvcZLmcfl_**
Document Elements	-	Additional nodes in document tree: #myId{visibility:visible}", "childNodes":[]}]},{"nodeType":3, "nodeName":"#text",...

Table 3. Deviations between headles Selenium+ChromeDriver and Chrome. The resulting fingerprint surface is marked in bold.

6 Looking for web bot detectors in the wild

In this section, we use the identified web bot fingerprint surfaces to develop a scanner that can detect web bot detectors. Since the fingerprint surfaces are limited to the web bots we tested, we extended our set of fingerprint surfaces with results from the reverse analysis and other common best fingerprinting-alike practices to detect web bots. The resulting fingerprint features were expressed as patterns, which were loaded into the scanner. The scanner is built on top of the OpenWPM web measurement framework [EN16]. OpenWPM facilitates the use of a full-fledged browser controllable via Selenium. The scanner thus resembles a regular browser and cannot be distinguished as a web bot easily

without client-side detection. Moreover, OpenWPM implements several means to provide stability and recovery routines for large-scale web measurement studies.

We set up the scanning process as follows: first, the scanner connects to a website's main page and retrieves all scripts that are included by *src* attributes. Each script that matches at least one pattern is stored in its original form, together with the matched patterns and website metadata. Scripts that do not trigger a match are discarded.

6.1 Design decisions

Some parts of the fingerprint surface concern not properties, but their values. For example, in Table 3, the value of `navigator.useragent` contains 'Headless-Chrome' for a web bot, instead of 'Chrome' for the regular browser. To detect whether client-side scripting checks for such values, we use static analysis. To perform static analysis, the detection must account for different character encodings, source code obfuscation and minified code. Therefore, the scanner transforms scripts to a supported encoding, removes comments and de-obfuscate hexadecimals. The resulting source code can be scanned for patterns pertaining to a specific web bot's fingerprint surface.

Note that our approach has several limitations. In the current setup, the scanner does not traverse websites. As such, data collection is limited to scripts included on the first page. We caution here once again that browser fingerprinting is only one of a handful of approaches to detecting bots. For example, this approach cannot detect behavioural detection. Nevertheless, from the reverse analysis we learned that browser fingerprinting by itself can be sufficient for a detector script to conclude that the visitor is a web bot, irrespective of the outcome of other detection methods. Finally, as a consequence of using static analysis, this approach will miss out on dynamically included scripts [NIK+12]. Thus, our approach will provide a lower bound on the prevalence of web bot detection in this respect.

6.2 Patterns to detect web bot detectors

To determine if a website is using web bot detection, we check whether it accesses the fingerprint surface. We do this by checking whether the client-side JavaScript of the website includes patterns that are unique to an individual bot's fingerprint surface. We derived these patterns as follows: firstly, from the the determined fingerprint surfaces, secondly, from the reverse analysis. With these we executed preliminary runs of the scanner, which resulted in more candidate scripts. The third source of patterns stems from new scripts identified in this stage. Table 4 lists the used patterns. Patterns derived from reverse analysis of the Distil bot detector are marked as 'RA', while patterns that emerged from the gathered fingerprint surfaces are marked as 'FP'. Finally, later identified web bot detector script are marked as 'RA2'. For all patterns where it is clear which web bot they detect, this is indicated in the table. By construction, this is the case for all fingerprint surface-derived patterns. However, not all patterns from the various

Pattern	*Source*	*Detects*
"webdriver",'webdriver', \.webdriver(?![a-zA-z-])	FP	IE + Sel. WD
__webdriver_script_fn	FP	IE + Sel. WD
_Selenium_IDE_Recorder	FP	FireFox Selenium IDE
PhantomJS(?![a-zA-z-])	FP	PhantomJS
_phantom(?![a-zA-z-])	FP	PhantomJS
callPhantom	FP	PhantomJS
HeadlessChrome	FP	Chrome + Chromium
__nightmare	FP	PhantomJS
__IE_DEVTOOLBAR_CONSOLE_EVAL_ERROR	FP	InternetExplorer
__IE_DEVTOOLBAR_CONSOLE_EVAL_ERRORCODE	FP	InternetExplorer
$cdc	FP	Chrome* WebDriver
webdriver_evaluate	RA	FX-Driver
fxdriver_evaluate	RA	Selenium
domAutomation	RA	Headless Chrome

Pattern	*Source*	*Detects*
_selenium	RA	?
selenium_evaluate	RA	?
webdriver_script_func	RA	?
selenium_unwrapped	RA	?
driver_unwrapped	RA	?
webdriver_unwrapped	RA	?
driver_evaluate	RA	?
fxdriver_unwrapped	RA	?
Sequentum	RA	?
callSelenium	RA	?
script_function	RA	?
$wdc	RA2	Selendroid
webdriver-evaluate	RA2	FX-Driver
domAutomationController	RA2	Headless Chrome
__phantomas(?![a-zA-z-])	RA2	PhantomJS
['PhantomJS(?![a-zA-z-])','botPattern']	RA2	PhantomJS

Chrome*: Chrome and Chromium.
FP: From fingerprint surface.
RA: From reverse analysis of the Distil script.
RA2: From the reverse analysis of later identified scripts.

Table 4. Web bot detector patterns derived from reverse analysis and fingerprint surface.

reverse analysed scripts could as readily be related to specific web bots. These are marked as '?' in the column *Detects* in Table 4.

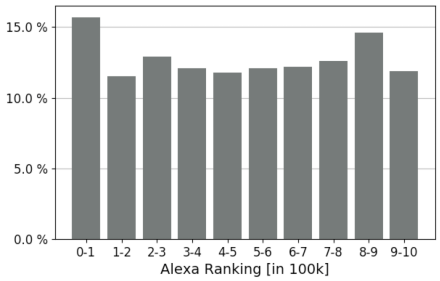

Fig. 2. Fraction of web bot detectors within the Alexa Top 1M.

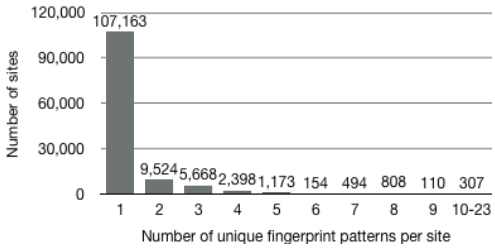

Fig. 3. Number of unique hits per website. Each pattern is counted once per site.

6.3 Results of a 1-Million scan

We deployed our scanner on the Alexa Top 1M and found 127,799 sites with scripts that match one or more of our patterns. Except for the Top 100K, these sites are mostly equally distributed. In the Top 100K, the amount of web bot detection (15.7K sites) is around a quarter higher than for the rest, which averages to 12.7K sites using detection per 100K sites (see the distribution in Figure 2).

Many of these sites employ PhantomJS-detection. In Table 5, we see that out of the 180,065 matches to the pattern list, the top three patterns were all

	Patterns	#sites	*Validation* sample size	FP	FP%
1	`PhantomJS(?![a-zA-z-])`	115,940	383	4	1%
2	`callPhantom`	11,759	373	0	0%
3	`_phantom(?![a-zA-z-])`	11,747	372	15	4%
4	`HeadlessChrome`	10,279	371	0	0%
5	`"webdriver", 'webdriver', \.webdriver(?![a-zA-z-])`	8,512	368	2	1%
6	`webdriver-evaluate`	5,441	all scripts	0	0%
7	`domAutomation`	2,238	328	0	0%
8	`__phantomas(?![a-zA-z-])`	2,123	317	0	0%
9	`domAutomationController`	1,852	all scripts	0	0%
10	`__webdriver_script_fn`	1,499	306	0	0%
11	`Sequentum`	1,479	all scripts	0	0%
12	`$cdc`	1,251	all scripts	19	2%
13	`$[a-z]dc_`	1,240	all scripts	8	1%
14	`_selenium`	636	240	86	36%
15	`_Selenium_IDE_Recorder`	339	all scripts	0	0%
16	`driver_unwrapped`	318	all scripts	0	0%
17	`fxdriver_unwrapped`	318	all scripts	0	0%
18	`driver_evaluate`	316	all scripts	0	0%
19	`webdriver_evaluate`	315	all scripts	0	0%
20	`selenium_unwrapped`	307	all scripts	0	0%
21	`webdriver_unwrapped`	307	all scripts	0	0%
22	`selenium_evaluate`	306	all scripts	0	0%
23	`__nightmare`	296	all scripts	0	0%
24	`callSelenium`	296	all scripts	0	0%
25	`webdriver_script_func`	295	all scripts	0	0%
26	`webdriver_script_function`	293	all scripts	0	0%
27	`fxdriver_evaluate`	267	all scripts	0	0%
28	`$wdc`	47	all scripts	20	43%
29	`PhantomJS(?![a-zA-z-]) && botPattern`	31	all scripts	0	0%

Table 5. Pattern matches within the Alexa Top 1M.

PhantomJS-related and together accounted for 139,446 hits. When all PhantomJS-related patterns are grouped, we find that 93.76% of the scripts in which we found web bot detection, contains one or more of these patterns.

While less prevalent, detection of other web bots does occur. The next most popular patterns are related to WebDriver (1.31% of sites in Alexa Top 1M), Selenium (1.34%), and Chrome in headless mode (0.99%). The other patterns were seldomly encountered, none of them on more than 0.2% of sites.

We also investigated how many different patterns occurred in detector scripts (Figure 3). Most sites only triggered one pattern. For 96% of the sites that only matched one pattern, the pattern was "`PhantomJS(?![a-zA-z-])`". This suggests that simple PhantomJS checks are relatively common, while actual bot detection using client-side detection is rare. The highest number of unique patterns found on a site was 23.

6.4 Validation

In order to validate the correctness of our results, we check if there are non-bot detectors among our collection of bot detector scripts, so called false positives. To confirm a script is a bot detector we perform code reviews. A script is marked as confirmed if it accesses unique web bot properties or values via the DOM. Some detectors separate their detection keywords in a different file, as we encountered

that during our reverse analysis in Section 3. Therefore, we also interpret these scripts (listing multiple of our patterns) as detectors. Note, our validation is limited to false positives. We do not investigate false negatives (scripts that do perform bot detection, but were not detected): such scripts were not collected.

In a preliminary validation run, we observed that some patterns are more likely to produce false positives than others. Therefore, we assessed false positive rates for the patterns individually by building sets containing all scripts that triggered a specific pattern.

Table 5 depicts the results of our validation by showing the set size of validated scripts, number of false positives (FP) and the percentage of false positives per set. For 20 out of 29 patterns, many sites used the exact same script. In these cases, we validated the entire set by reviewing all unique scripts in the set. Any found false positives were weighted accordingly.

For the remaining patterns, the sets of scripts was too diverse to allow full manual validation. We used random sampling with a sample size that provides 95% confidence to validate these patterns. In Table 5, we list these patterns together with the sample size.

Our validation shows that the patterns *$wdc* and *_selenium* raise a non-negligible number of false positives, though scripts matching these patterns only constitute a tiny portion of our dataset. The other patterns are good indicators of web bot detection.

7 Cloaking: are some browsers more equal than others?

Finally, we studied whether sites we identified as engaging in bot detection respond differently to web bot visitors. That is: do these websites tailor their response to specific web bots? Note that we manually examine the response to generic web bots, which differs from previous work that investigated cloaking in the context of search engine manipulation [ITK+16, WD05, WSV11].

We first assess the range of visible variations by visiting 20 sites that triggered a high number of patterns. To do so, we visited websites and took screenshots with a manual driven Chrome browser and an automated PhantomJS browser – the most detected and most detectable automation framework in our study. This was repeated 5 times to exclude other causes, such as content updates. We found four types of deviating responses: CAPTCHAs (3 sites), error responses of being blocked (1 site), connection cancellation or content not displayed (1 site) and different content (12 sites).

The differences in content concerned page layout (2 sites), videos that do not load (3 sites), missing ads (9 sites) and missing elements (1 site). We found that these deviations are highly likely to be caused by bot detection, e.g. one site in our set does not display login elements to web bots (see Figure 6 in Appendix A). In contrast, deviations such as malformed page layouts may be a result of PhantomJS' rendering engine.

We found that sites with missing videos use scripts to serve the videos by wistia.com. These scripts include code to detect PhantomJS. We therefore be-

lieve the lack of video to be due to web bot detection, though we cannot be certain without reverse engineering these scripts fully.

Lastly, we explored how often deviations due to bot detection occur. In addition to PhantomJS, we add a Selenium-driven Chrome browser. We randomly selected 108 sites out of our set of detectors. Each site was visited once manually and 5 times with each bot, using different machines and IPs. By comparing the resulting screenshots we found deviations on 50 sites. From these deviations, we removed every observation (e.g. deformed layouts and inconsistent results over multiple visits) that we could not clearly relate to web bot detection. This results in 29 websites where we interpret deviations as a cause of web bot detection (see Appendix A).

We found 10 websites that do not display the main page or show error messages to web bots. CAPTCHAs were shown on 2 sites. We further encountered missing elements on 2 sites and videos failed to load on 4 sites. Lastly, 15 sites served less ads. Overall, deviations appeared more often in PhantomJS (24) than in Selenium-driven Chrome browsers (14).

8 Conclusions

The detection of web bots is crucial to protect websites against malicious bots. At the same time, it affects automated measurements of the web. This raises the question of how reliable such measurements are. Determining how many websites use web bot detection puts an upper bound on how many websites may respond differently to web bots than to regular browsers.

This study explored how prevalent client-side web bot detection is. We reverse engineered a commercial client-side web bot detection script, and found that it partially relied on browser fingerprinting. Leveraging this finding, we set out to determine the unique parts of the browser fingerprint of various web bots: their *fingerprint surface*. To this end, we grouped browsers into families as determined by their layout and rendering engines. Differences between members of the same family then constituted the fingerprint surface. We determined the fingerprint surface of 14 web bots. We found PhantomJS in particular to stand out: it has many features by which it can be detected.

We translated the fingerprint surfaces into patterns to look for in JavaScript source code, and added additional patterns from the reverse analysis and common best practices. We then developed a scanner built upon OpenWPM to scan the JavaScript source of the main page of all websites in the Alexa Top 1M. We found that over 12% of websites detect PhantomJS. Other web bots are detected less frequently, but browser automation frameworks Selenium, WebDriver and Chrome Headless are each detected on about 1% of the sites.

Lastly, we performed a qualitative investigation whether web bot detection leads to a different web page. We found that indeed, some browsers are more equal than others: CAPTCHAs, blocked responses different content occur. In a further experiment, we attribute at least 29 out of 108 encountered differences.

Future work. We plan to investigate advanced fingerprint techniques to reveal further unique properties in web bots fingerprint surfaces. Further, expanding our current measurement technique with dynamic approaches can contribute to deliver more accurate measurements of the occurrence of web bot detection. Finally, we plan to develop a stealth scanner, whose browser fingerprint is as close as possible to that of a regular browser. This can be used for future studies, as well as experiments repeating previous studies to determine the effect of web bot detection on those studies.

References

AJN+13. Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.

BFGI11. K. Boda, Á. M. Földes, G. György Gulyás, and S. Imre. User tracking on the web via cross-browser fingerprinting. In *Proc. 16th Nordic Conference on Information Security Technology for Applications (NordSec'11)*, volume 7161 of *LNCS*, pages 31–46. Springer-Verlag, 2011.

BLRP10. Douglas Brewer, Kang Li, Lakshmish Ramaswamy, and Calton Pu. A link obfuscation service to detect webbots. In *2010 IEEE International Conference on Services Computing, SCC 2010, Miami, Florida, USA, July 5-10, 2010*, pages 433–440, 2010.

CGK+13. Zi Chu, Steven Gianvecchio, Aaron Koehl, Haining Wang, and Sushil Jajodia. Blog or block: Detecting blog bots through behavioral biometrics. *Computer Networks*, 57(3):634–646, 2013.

DG11. Derek Doran and Swapna S. Gokhale. Web robot detection techniques: overview and limitations. *Data Min. Knowl. Discov.*, 22(1-2):183–210, 2011.

Eck10. Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th Privacy Enhancing Technologies Symposium, (PETS'10)*, volume 6205 of *LNCS*, pages 1–18. Springer, 2010.

EN16. Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1388–1401. ACM, 2016.

GG05a. Alan Grosskurth and Michael W. Godfrey. A reference architecture for web browsers. In *21st IEEE International Conference on Software Maintenance (ICSM 2005), 25-30 September 2005, Budapest, Hungary*, pages 661–664, 2005.

GG05b. Zoltán Gyöngyi and Hector Garcia-Molina. Web spam taxonomy. In *AIRWeb 2005, First International Workshop on Adversarial Information Retrieval on the Web, co-located with the WWW conference, Chiba, Japan, May 2005*, pages 39–47, 2005.

ITK+16. Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean Michel Picod, and Elie Bursztein. Cloak of visibility: Detecting when machines browse a different web. In *Proceedings of the 37th IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 743–758, 2016.

MBYS11. K. Mowery, D. Bogenreif, S. Yilek, and H. Shacham. Fingerprinting information in JavaScript implementations. In *Proc. Web 2.0 Security & Privacy (W2SP'11)*, volume 2. IEEE Computer Society, 2011.

MS12. K. Mowery and H. Shacham. Pixel Perfect: Fingerprinting canvas in HTML5. In *Proc. Web 2.0 Security & Privacy (W2SP'12)*. IEEE Computer Society, 2012.

NIK+12. Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 19th ACM Conference on Computer and Communications Security, CCS 2012, Raleigh, NC, USA, October 16-18, 2012*, pages 736–747, 2012.

NKJ+13. Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proc. 34th IEEE Symposium on Security and Privacy (SP'13)*, pages 541–555. IEEE Computer Society, 2013.

PPLC06. KyoungSoo Park, Vivek S. Pai, Kang-Won Lee, and Seraphin B. Calo. Securing web service by automatic robot detection. In *Proceedings of the 2006 USENIX Annual Technical Conference, Boston, MA, USA, May 30 - June 3, 2006*, pages 255–260, 2006.

PSF16. Kien Pham, Aécio S. R. Santos, and Juliana Freire. Understanding website behavior based on user agent. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*, pages 1053–1056, 2016.

SD09. Athena Stassopoulou and Marios D. Dikaiakos. Web robot detection: A probabilistic reasoning approach. *Computer Networks*, 53(3):265–278, 2009.

TJM15. Christof Ferreira Torres, Hugo L. Jonker, and Sjouke Mauw. Fp-block: Usable web privacy by controlling browser fingerprinting. In *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS'15), Proceedings, Part II*, volume 9327 of *LNCS*, pages 3–19. Springer, 2015.

vABHL03. Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: using hard AI problems for security. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 294–311, 2003.

VYG13. Shardul Vikram, Chao Yang, and Guofei Gu. NOMAD: towards non-intrusive moving-target defense against web bots. In *IEEE Conference on Communications and Network Security, CNS 2013, National Harbor, MD, USA, October 14-16, 2013*, pages 55–63, 2013.

WD05. Baoning Wu and Brian D. Davison. Cloaking and redirection: A preliminary study. In *AIRWeb 2005, First International Workshop on Adversarial Information Retrieval on the Web, co-located with the WWW conference, Chiba, Japan, May 2005*, pages 7–16, 2005.

WSV11. David Y. Wang, Stefan Savage, and Geoffrey M. Voelker. Cloak and dagger: dynamics of web search cloaking. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 477–490, 2011.

XLC+18. Haitao Xu, Zhao Li, Chen Chu, Yuanmi Chen, Yifan Yang, Haifeng Lu, Haining Wang, and Angelos Stavrou. Detecting and characterizing web bot traffic in a large e-commerce marketplace. In *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II*, pages 143–163, 2018.

A Screenshots from effects of web bot detection

Comparing websites requested from web bots with websites requested from human controlled browsers can lead to various deviations. On sites that perform bot detection, we found websites that do not display login elements for visitors using PhantomJS or do not load videos (c.f. Figure 6 and Figure 7).

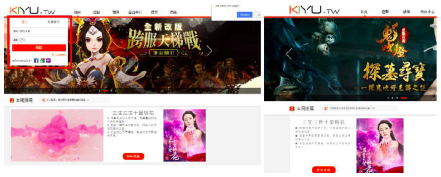

Fig. 4. Missing login fields on kiyu.tw.

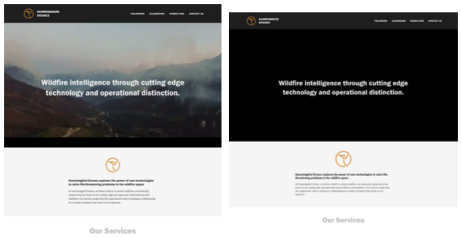

Fig. 5. Missing video on hummingbirddrones.ca.

Fig. 6. Blockage and loading of a CAPTCHA on frankmotorsinc.com.

Fig. 7. Missing ads on cordcuttersnews.com.

B Advanced notes to determining the fingerprint surface

The following subsections provide further insights into our process to derive a fingerprint surface for web bots. We begin with the description of our modification

to fingerprintjs2, in order to cover more web bot specific characteristics. Then, we give an overview of our used setup during to make this process repeatable.

B.1 Extra elements used in determining the fingerprint

There are several discussions on best practices for identifying web bots available online. From this, we included the following extra elements to include in the browser fingerprint:

- Lack of "bind" JavaScript engine feature[11].
  Certain older web bots make use of outdated JavaScript engines that do not support this feature, which allows them to be distinguished from full JavaScript engines.
- StackTrace[12].
  When throwing an error in PhantomJS, the resulting StackTrace includes the string 'phantomjs'.
- Properties of missing images[13].
  The width and height of a missing image is zero in headless Chrome, while being non-zero in full Chrome.
- Sandboxed `XMLHttpRequest`[11].
  PhantomJS allows turning off "web-security", which permits a website to execute a cross-domain `XMLHttpRequest()`.
- Autoclosing dialog windows[11].
  PhantomJS auto-closes dialog windows.

B.2 Setup for determining the fingerprint surface

The resulting fingerprint surface of a web bot framework depends on used versions of the framework and corresponding browser. The versions and setup that were used during our experiment are listed in Table 6. Human-controlled browsers are marked as bold.

[11] https://blog.shapesecurity.com/2015/01/22/detecting-phantomjs-based-visitors/
[12] https://www.slideshare.net/SergeyShekyan/shekyan-zhang-owasp
[13] https://antoinevastel.com/bot%20detection/2017/08/05/detect-chrome-headless.html

OS	*Browser*	*Version*
Ubuntu 16.04	**Chrome**	64.0.3282.140
	Chrome & Chromium + Selenium ChromeDriver	Sel: 4.0.0; WD: 2.35
	Chrome + Selenium IDE	C: 62.0.3202.94; IDE: 3.0.1
	NightmareJS	2.10
	Chrome & Chromium Puppeteer	1.1.0
	Firefox	54.0.1
	Firefox + Selenium GeckoDriver	Sel: 4.0.0; WD: 0.19.1
	Firefox + Selenium IDE	2.9.1 & 3.0.1
	Opera	53
	Opera + Selenium OperaDriver	Sel: 4.0.0; WD: 2.36
Windows 10	**Microsoft Edge**	41.16.299.15.0
	Microsoft Edge + Selenium Edge Driver	Sel: 4.0.1; WD: 10.0.16299.15
	Internet Explorer	11.0.50
	IE + Selenium InternetExplorerWebDriver	Sel: 4.0.1; WD: 3.9.0
OS X 10.13.5	**Safari**	12.0
	SafariDriver	Sel: 3.6.0; WD: N/A
	PhantomJS	2.1.0

Table 6. Configurations used to determine fingerprint surfaces.