Bachelor thesis
Computing Science

Radboud University

Recognising Client-side Behavioral Detection of Web Bots

Author:
Mitchel Jansen
s4810732

First assessor/supervisor:
Dr. Erik Poll
erik.poll@cs.ru.nl

Second assessor:
Dr. Ir. Hugo Jonker
hugo.jonker@ou.nl

January 9, 2021

Abstract

To prevent data stealing, many websites use techniques to detect automated visitors. Collectively, such techniques are called web bot detection. There are, generally speaking, two types of such techniques: (1) fingerprinting techniques, and (2) behavioral detection. Behavioral detection is an upcoming bot detection method which uses different techniques than those considered in browser fingerprinting. While there have been several studies into fingerprint-based detection of web bots, there has been comparatively little investigation of behavioral-based detection. In this thesis, we investigate behavioral detection and its countermeasures. Moreover, we determine whether this is simply an extension of browser fingerprinting or if its use extends beyond browser fingerprinting.

The goal of this work is to expand the identification approach by Jonker et al. to cover dynamic aspects of web bot detection, specifically: behavioral detection of web bots. A key subgoal of this project is therefore to uncover when current real-world bot detection methods deem evidence sufficient to classify a visitor as a bot.

We achieve this goal through the following contributions. We determine a list of known methods for behavioral detection, based on literature and a practical investigation. Moreover, we design a method to determine whether a script is using behavioral detection, given the detection methods we have found in our manual reverse analysis. Besides that, we implement this approach in a static HTML scanner and scan the homepages of the Tranco Top 10,000. Ultimately, we determine the false positive rate of our scanner by manual analysis of a statistical significant number of scripts in each category using random sampling.

First and foremost, bot detection scripts will be reverse-analyzed manually to acquire code that is used for behavioral detection. Afterwards, this manual process will be automated, utilizing the code snippets and its features which were found manually. Processing and categorizing this data can then commence. Through analyzing the categorized data, conclusions can be drawn regarding which code is utilized for behavioral detection of web bots.

It is important to note that we first need to test whether the approach taken in this research is appropriate and applicable to behavioral detection the same way it is applicable to fingerprinting techniques. Consequently, we need to find out why it is more difficult to recognise behavioral detection than fingerprinting techniques.

We found that 17,203 out of the 17,862 scripts have between 0% and 20% chance of containing behavioral-based detection. Thus, we can conclude with certainty that 17,203 scripts do not use behavioral-based detection if we disregard any false negatives. 290 scripts have between 20% and 40% chance of containing behavioral detection. Meaning, 290 scripts possibly have some small form of behavioral-based detection, but have such a low chance that we believe no behavioral detection will be present. 97 scripts are in a range where we simply do not know whether they carry out behavioral detection until closer inspection as they do contain a limited amount of behavioral detection methods. However, 69 scripts have a good chance of containing behavioral-based detection since they contain multiple behavioral detection methods and the 23

scripts with the highest chance of containing behavioral detection certainly do contain behavioral detection.

In this research we were able to gather significant results using a static scraping tool. Thus, this research is the basis for further research investigating behavioral detection using more advanced techniques such as machine learning with a neural network and a dynamic scraping tool. In short, this research gives all of the ingredients that can be used in machine learning as well as the properties that a dynamic scraper needs to possess.

Contents

Code snippet

Chapter 1

Introduction

There are various ways to automatically gather information from the web, such as web spiders, scrapers, etc. In this study, we refer to such automated visitors of a web page as web bots. Web bots are widely used to automatically gather information from the web. Web bots can be used maliciously, stealing information, but they may also be used benignly, such as using the information for research purposes. Web sites can try to prevent malicious web bots from accessing their content. To this end, they must first identify the visitor as a scraper. Jonker et al. [JKV19] investigated identifying web bots based on subtle differences between their browser fingerprint and the browser fingerprint of regular browsers. Browser fingerprinting focuses on static properties of a web bot.

The goal of this research is to expand the identification approach by Jonker et al. to cover dynamic aspects of web bot detection, specifically: behavioral detection of web bots. Do note that while static differences between browser fingerprints can prove the visitor is a web bot, differences in behavior can only provide circumstantial evidence. Where identifying detection techniques based on browser fingerprinting is deterministic (true/false), recognising behavioral detection is inherently probabilistic. A key sub-goal of this project is therefore to uncover when current real-world bot detection methods deem evidence sufficient to classify a visitor as a bot.

Most will know behavioral detection through the well-known and more-or-less accepted price tracking sites. However, the usage of behavioral detection has benign uses such as research as well as nefarious uses such as content stealing. Therefore, countermeasures ideally only affect bots. However, to acquire this we need to make sure that the countermeasures can distinguish bots from humans.

There are two ways to make such a distinction. The first option is to find static aspects that differ between bots and humans, called properties. The other option is to find dynamic aspects that different between bots and humans, called behavior. The differences which originate from static aspects. such as browser fingerprint, have already been explored (see e.g. Jonker et al. [JKV19]). Dynamic aspects, such as behavior, seem to be used in practise as well, but comparatively few studies have looked into this.

Behavioral detection focuses on a web bots behavior, i.e., the way it interacts with the visited web page. This may include measurements of

– mouse behavior, including mouse movement and mouse clicks;

– typing behavior, including key presses;
– timing between subsequent requests;
– scrolling.

1.1 Objectives and research questions

The main goal of this project is to investigate how web sites currently employ behavioral detection. Subsequently, we can derive what countermeasures should be designed to prevent behavioral detection in practice.

The thesis should give all features necessary for machine learning so these can be applied in the future. It is important to note that creating a classifier for this research was possible, but was not used since we first need to identify whether it makes sense to train a classifier. Besides that, it is difficult to acquire enough training data regarding behavioral detection for machine learning to work to its full capacity.

The above-mentioned goals have two research topics that need to be addressed. These are both contained in the overarching main research question:

How is behavioral detection currently being used on the internet?

The main research question is divided into the following sub-questions:

Sub-question 1: Which methods for behavioral detection exist?
Behavioral detection is a relatively new detection method for bots. Some research has already been done on the different methods for behavioral detection. However, the usage of these different methods is still unknown. Therefore we will be reverse-analyzing scripts and doing a literature study to determine the different methods of behavioral detection used in practise.

Sub-question 2: How can behavioral detection be recognised?
In the case of fingerprint-based detection, some properties are unique to a web bot. Accessing these is thus only relevant for identifying a bot [JKV19]. However, in the case of behavioral detection we cannot say the same. Methods employed in behavioral detection may also be used for other purposes. For example, tracking mouse movement may be used for behavioral detection, or to facilitate an online drawing program. Thus, presence of a certain technique which can also be used for behavioral detection does not necessarily imply the site is using behavioral detection. This sub-question addresses this point: when is the use of techniques a clear indicator of behavioral detection? The answer to this question shows us how to recognise behavioral detection.

Sub-question 3: How many sites are using behavioral detection?
This sub-question investigates the prevalence of behavioral detection. Consequently, the answer to this question indicates how relevant behavioral detection is currently when it comes to bot detection as a whole.

This thesis and its accompanying study will answer these research questions with the following contributions:

- We determine a list of known methods for behavioral detection, based on literature and a practical investigation.
  - We establish a list of current behavioral detection methods used in practice, based on methods found in literature combined with results from the reverse analysis.
  - We manually reverse engineer several known bot-detection scripts to analyze their approach to behavioral detection.
- We design a method to determine if a script is using behavioral detection given the discussed detection methods. Our approach is based on Vlot's scoring approach [Vlo19]. Unlike Vlot's fingerprint-based approach, behavioral detection cannot be identified with 100% certainty. This necessitates a novel, probabilistic approach to recognising behavioral detection.
- We implement this approach in a static HTML scanner and scan the home pages of the Tranco[1] Top 10,000 websites for behavioral detection, on which the scanner found 17,862 scripts. Our scanner found 23 scripts with a score of 100%, 69 scripts between 60–95%), 97 scripts between 40–60%, 290 scripts between 20%–40% and 17,203 scripts that did not use any known behavioral detection method (scoring 0%–20%).
- We determine the false positive rate of our scanner by manual analysis of a statistical significant number of scripts in each category using random sampling.

[1] https://tranco-list.eu/

Chapter 2

Related Work

Recent studies into web bot detection mainly focus on browser fingerprinting. There are only a few recent studies regarding behavioral detection of web bots.

2.1 Bot detection

Bot detection is the overarching name for all different methods concerning the detection of bots on the web. The research regarding bot detection gives insight in the way bot detection takes place in general and serves as a base for further research.

According to Ji et al. bot detection can be categorized into only two categories, host-based and network-based [JHZ+14]. Network-based bot detection monitors network packets for signs of bots, while host-based detection monitors the objects and processes of the system. In addition, Ji et al. theorize that behavior-based methods are more practical and effective for single process bots, but become less effective when detecting multi-process bots. Multi-process bots have two specific features. As opposed to a single process bot, they can separate command and control connections from malicious behavior and they are able to assign malicious behavior to multiple processes. We will not be making this distinction in our research.

Taking a closer look at the bot detection itself, Dolfing [Dol19] has done a study on multiple papers discussing browser fingerprinting. He mentions that different terms are used across these different papers and he wants to make sure the the reasoning is aligned between the different findings. This study adds to the research regarding browser fingerprinting by clearing up different studies and therefore making sure that future research can focus better on the aspects they want to discuss. In our thesis, we want to use this same approach regarding the behavioral detection methods. As of right now there are multiple papers discussing one or more behavioral detection method(s). However, a paper which contains a list of behavioral detection methods is not available to our knowledge.

Behavioral detection methods can have a major impact on scientific research by limiting automating scraping of data even though it is used for benign purposes. Englehardt et al. [EN16] created OpenWPM, a web bot framework intended for scientific studies into measuring online privacy. OpenWPM has several anti-bot-detection features, that focus exclusively on behavioral detection. OpenWPM has been successful, in that it has been used in over 50 studies to date.

Using a scanner built on top of this OpenWPM framework, Jonker et al. [JKV19] studied browser-fingerprinting-based bot detection. They manually analysed several known bot-detection scripts and found they partially relied on browser fingerprinting. They extrapolated from there and tested the browser fingerprints of several popular web bot frameworks against the browser on which they are based. They found that these frameworks introduce subtle differences in the browser fingerprint. In a study of the Alexa Top 1 Million, they found that 12.7% of websites used some form of browser-fingerprinting-based web bot detection.

Regarding detection and analysis, multiple studies have been conducted. Tan et al. [TK02] have proposed navigational pattern analysis. This was a new approach to bot detection, focusing on how the client navigates the site. It is an early form of behavioral detection that can be executed fully server-side.

Besides that, Schwarz et al. [SLG19] have conducted a study on JavaScript Template Attacks. These attacks are automated and detect differences between browser engines based on their environment. The detection of these differences between the browser engines applies to the browser fingerprinting to distinguish bots from humans.

2.2 Behavioral detection

Vlot investigated detection of web bots using browser fingerprinting [Vlo19]. He applies the concept of *fingerprinting surface* to several web bots and finds, for each web bot, several attributes in which their browser fingerprint is distinct from other browsers within the same browser family. Based on these findings, he built a scanner to detect whether websites were accessing these unique parts of the browser fingerprint. We will adapt his approach to identifying fingerprint-based bot detection to be applicable to behavioral detection.

The work of Bai et al. [BXZH14] focuses on bogus behavior in web crawler measurement. Their analysis features looking at the behavior of web crawler measurement. However, in this research they propose a model which focuses on the traffic of the crawlers at the network gateway of a network operator. We are taking a different approach with this research, by investigating behavioral detection, using pattern matching instead of investigating the network traffic generated.

Most behavioral detection methods can be used by any company, whereas other approaches give a website-specific bot-detection countermeasure. One of these approaches has been explored by Haidar et al. [HE17]. They proposed to use a site-specific classifier to distinguish between humans and bots, based on the path traversal of the website. They trained their classifier on input from genuine human visitors. In their subsequent test, the classifier detected bots after these had, on average, visited 10 pages. This approach may thus be valuable to websites seeking to prevent wholesale scraping of their content, but is not useful against specific scrapers that only load a handful of pages.

Iliou et al. have done research towards creating a framework to detect advanced bots [IKT+19]. In this research they have assessed HTTP traffic and noticed that the accuracy is high whenever the bots do not try and hide their identity. They consider advanced web bots as web bots that present a browser fingerprint and possibly contain human-like behavior. A research and framework such as the one investigated by Iliou

et al. is built upon to try and detect these more advanced bots simulating human-like behavior.

Chapter 3

Methodology

This research will look into behavioral detection and how this amounts to bot detection as a whole. First and foremost, bot detection scripts will be reverse-analyzed manually to acquire code that is used for behavioral detection. Afterwards, this manual process will be automated, utilizing the code found manually. Processing and categorizing this data can then commence. Through analyzing the categorized data, conclusions can be drawn regarding which code is utilized for detecting bots.

Whereas reverse-analysis has given insight into the way in which scripts and their corresponding code are responsible for detection, this does not give information regarding the behavior of such a script. Therefore, the next step in this research is performing action-based analysis. Instead of analyzing the scripts, it is now time to start testing these scripts in action. First of all, these scripts should be loaded and specific functions should be called to check whether behavior of the user is tracked. Initially this can and should be done locally to check the behavioral detection and responses to user behavior which the scripts return. Afterwards, these same tests should be executed on a web server and it should be investigated whether any of the behavior alters.

Whenever alternate behavior has been detected, this needs to be investigated further and changes to testing might need to be implemented. If not, the behavioral detection scripts perform the same both locally and on a web server. Since they work the same, tests can and should be performed both locally and on the web server so that consistency is maintained. Immediately upon receiving feedback from the behavioral detection scripts, the main task becomes collecting this feedback and storing it accordingly. After collecting a sufficient amount of data, it should be clear how the bot detection scripts have determined that the user is a regular user and not a bot.

Along with behavioral detection regarding a regular user, it is now important to repeat this same process and acquire information using a bot. This bot should trigger the behavioral detection scripts and data collection should commence. After this elongated process it is time to compare the collected data with the data collected on a regular user.

The complicating factor is the fact that the technique of detecting bots can be 'perfect', but the technique to detect the user of these 'perfect' techniques cannot exclude false positives. In short, it is difficult to distinguish between bot-detection and benign use of behavioral detection techniques. As mentioned before, since countermeasures can lead to websites masking their content, scraping studies can become unreliable.

Therefore, scraping studies should detect whether web sites use detection techniques. For fingerprint-based techniques there is already an approach. However, for behavioral-detection based techniques there is no approach known to us. This is why we have looked into creating an approach for behavioral-detection based techniques.

Chapter 4

Known methods for behavioral detection

This chapter serves to show the different behavioral detection methods and their countermeasures discussed in literature. The methods and their accompanying brief explanation are followed by the countermeasures. Besides that, the method and its countermeasures are followed by papers in which the detection method is discussed. Lastly, the method and countermeasure can be followed by a remark. The detection method, its countermeasure, the corresponding paper and the usefulness of the detection method are then also depicted in a table. Further explanation of the detection methods, its countermeasures and code examples of all of the detection methods below can be found in appendix B.

In Table 4.1, the overview of behavioral detection methods discussed in literature has been depicted. The detection methods are listed from a low sophistication level to a high sophistication level.

As shown in the table, each of the detection methods have been classified as either client-side or server-side detectable.

The detection method 'rate limiting of the user requests' can be done both client-side and server-side. In the code snippet B.0.3 we can see an example of the client-side detection method of rate limiting. Server-side rate limiting can be done by checking the logs.

The detection method 'honeypot trap' is done both server-side and client-side. This method checks whether the invisible element has been accessed. Server-side this can be found in the logs and client-side detection can trigger a response immediately after the element has been accessed.

The 'Captcha/Recaptcha' detection method is a client-side and server-side detection method. In this case a client-side verification of the Captcha/Recaptcha is not sufficient. For example, in cases where a user runs the form without JavaScript enabled, the verification will keep running and will keep sending submissions to the server. Therefore we need a server-side method which tests whether the Captcha/Recaptcha was run and has been completed.

'Site traversal' as a detection method is detected at the server-side. The server logs show the navigational data of the users that access the website.

The 'mouse movement' detection method requires both client-side and server-side

processing. Capturing the mouse movement is done client-side. This data is then sent from client-side to server-side for analysis.

Detection method	**Countermeasure**	**Discussed by**	**Detected at**
Rate limiting of user requests	Delays and random waits	[BMV20, MD11]	client or server
Honeypot trap	Do not access invisible elements	[SA18]	client and server
Captcha/Recaptcha	– Audio-version → speech-to-text – Crowdsourcing	[SA18, SKSP17, BMV20]	client and server
Site traversal	Human-like path traversal configuration	a.o. [TCK09, SVA11]	server
Mouse movement	Random mouse movements	[PB04, CGK+12]	client and server

Table 4.1: Overview of behavioral detection methods discussed in literature

Below we briefly describe the detection measures listed in Table 4.1.

- **Rate limiting:** as the name suggests, the amount of requests in a certain time-frame is limited.
  Remarks: rate limiting can be done both client-side and server-side. We will be focusing on client-side rate limiting.
- **Honeypot trap:** elements that cannot be accessed by a human user physically, but can be accessed by a bot, trigger a response whenever they are accessed.
  Remarks: with a honeypot trap, the trap itself is not considered a detection method. The detection part is checking whether a bot accesses the trap. This check is often done server-side and therefore we are unable to do anything with this information. However, we are able to detect honeypot traps client-side and therefore can circumvent the trap.
- **CAPTCHA/ReCAPTCHA:** whenever the CAPTCHA is solved, the system acknowledges you are a human user and you are able to continue traversing the website, otherwise you are blocked from the site.
- **Site traversal:** server-side logs contain data on the user navigation patterns of a site.
- **Mouse movement:** distinguish between human-like mouse movements and bot-like mouse movements.

How often do these methods occur in practise?

Testing how often these methods occur in practise, we have iterated over the 17,682 scripts resulting from collecting the scripts off of the homepages of the Tranco top 10,000.

Method	**Amount of scripts**	**Percentage of scripts**
Rate limiting	252	1.4%
Honeypot trap	81	0.5%
CAPTCHA/reCAPTCHA	2099	11.9%
Site traversal	335	1.9%
Mouse movement	9902	56.0%

Table 4.2: Known methods occurrences

Checking for all of the methods above was done by going through the scripts line by line and checking for methods and libraries known to be used when it comes to the method in question.

The table below shows which methods, libraries, links or classes were used to check whether the known behavioral detection method was present in the script.

Detection method	**Typical keywords**
Rate limiting	bottleneck
	express-brute
	request-rate-limiter
	smart-request-balancer
	ratelimit
	ratelimiter
Honeypot trap	honeypot
	spamtrap
	norobot
	nohuman
CAPTCHA/reCAPTCHA	src=”https://www.google.com/recaptcha/api.js”
	src=”https://www.google.com/recaptcha/api.js?...
	...onload=onloadCallback&render=explicit”
	class=”g-recaptcha”
	CAPTCHA
	reCAPTCHA
Site traversal	sessionStorage
	clickstream
Mouse movement	mousemove
	onmousemove
	mouseoverlistener
	DOMMousescroll

Table 4.3: List to check for known detection method

Important is to realize that when it comes to rate limiting there are methods such as `await` , but these cannot be checked using our analysis tool as this would return too many false positives since this is also often used outside rate limiting. With the honeypot trap, the same effect as the methods in the table can be realized by using absolute positioning outside of the scope of a human visitor. However, we cannot make the conclusion that absolute positioning outside of the canvas is automatically a honeypot trap.

Chapter 5

Reverse analysis of bot detection scripts

In chapter 4 we have seen which behavioral detection methods were documented in various papers. In this chapter we will focus on identifying which methods are present in various scripts. An approach to the reverse analysis will be discussed with its accompanying methods to identify the known behavioral detection. Lastly, the results of the reverse analysis will be documented and discussed.

5.1 Approach to reverse analysis

The initial step to our research is gaining access to data regarding behavioral detection. This part of the research is based on the data collected by Jonker et al. where bot detection scripts were obtained by using a scanner built on top of the OpenWPM web measurement framework [JKV19]. To study behavioral detection methods and especially how behavioral detection methods are implemented, were the initial focus. Through reverse-analysis of the scripts, insight was gained regarding this implementation.

During the data study, there were some problems which needed to be accounted for. Namely, methods were used to make analysis of the files more difficult. The process of analysing a file starts with de-minification of the code to make it more readable. Then beautification of the code is performed. The beautification tries to change the variable names back to the original names and makes sure the code layout is restored. After this is done, the manual investigation can commence.

An example of the de-minification and beautification process is shown below.

```
1 track:function(t,n){n.target=t,e.iframeTracker.handlersList.push(n),e(t).
      bind("mouseover",{handler:n},e.iframeTracker.mouseoverListener).bind("
      mouseout",{handler:n},e.iframeTracker.mouseoutListener)}
```

Code snippet 5.1: Original code

```
track: function(t, n) {
     n.target = t, e.iframeTracker.handlersList.push(n), e(t).bind("
         mouseover", {
       handler: n
     }, e.iframeTracker.mouseoverListener).bind("mouseout", {
       handler: n
     }, e.iframeTracker.mouseoutListener)
}
```

Code snippet 5.2: After de-minification process

```
track : function(source, options) {
    /** @type {!Object} */
    options.target = source;
    $.iframeTracker.handlersList.push(options);
    $(source).bind("mouseover", {
      handler : options
    }, $.iframeTracker.mouseoverListener).bind("mouseout", {
      handler : options
    }, $.iframeTracker.mouseoutListener);
}
```

Code snippet 5.3: After beautification process

Then, a manual analysis was done on the de-minified and beautified code. The results of the manual analysis showed that certain methods such as tracking mouse movement were not only used for behavioral detection, but also used for regular website operations. Thus, reinforcing the idea of possible false positives. These false positives were largely present as almost all scripts would have these methods applied.

During the manual reverse analysis we found that for a lot of behavioral detection scripts, API calls were being made. Since we do a static reverse analysis we ignore those API calls and focus on the methods we find in the source code. However, if further research commences regarding this topic this is worth to look into further. For example, finding out which behavioral detection method is present in one of the scripts and tracing this method through the API call with dynamic sink and source tracing.

The amount of bot detection scripts in the current database which were subject to the automated detection script was 19,393. The way in which these scripts were gathered structure-wise should be no problem for the data parsing which is done by the automated script. To make sure any updates on the database do not interfere with the automatic script, the database structure should be maintained. Since all bot detection scripts are distributed across more than 9000 folders, scripts are first stored in one directory. Then, the automatic script is able to iterate over the bot detection scripts and categorize the patterns in directories according to the behavioral detection methods table below.

5.2 Identifying known behavioral detection

The re-identification of known behavioral detection in this research was done in two ways. The usage of pattern matching has been used to compare event types and methods that attach events to specific elements for behavioral detection. The other form

of re-identification was done by methods that we will call "re-identification patterns". Whereas the pattern matching can lead to false positives, the re-identification patterns are used to take away these false positives. By checking the patterns used by companies that we know use behavioral detection against the patterns that might contain behavioral detection, we are able to confirm whether the patterns that we found to be possible behavioral detection are indeed behavioral detection.

In regards to the minification and obfuscation of code, we have not ran into problems. As we can see in code snippet 5.1, the code that is related to behavioral detection stays intact and has not been obfuscated. Therefore we do not need a de-obfuscation method when it comes to checking for behavioral detection patterns.

5.3 Results

In the reverse analysis 19,393 known bot detection scripts have been analysed. In the reverse analysis we have encountered the behavioral detection methods in table 5.1. These behavioral detection methods have unique properties which we can use to classify them. First of all, a lot of behavioral detection methods we have encountered can be categorised as 'event types'. An event in Java is triggered whenever a change in the graphical user interface occurs, making it perfect for behavioral detection. This can be used, for example, to check whether certain buttons are interacted with.

Besides the event types, we also came across methods which attach events to specific element (e.g. addEventListener). We categorise these methods as 'event-element-connections'. For each touch event type an event handler has to be registered. These touch events are mostly keyboard and mouse usage. Meaning, these patterns will most likely occur more frequently and can also frequently occur whenever no behavioral detection method is present.

Furthermore, there are unique patterns which identify a script or a host. These patterns highlight the typical style of a script and we can use these patterns to detect previously known and unknown scripts by re-identification. These 're-identification patterns' are used to check for strings and URLs which are linked to third parties known for behavioral detection. Meaning, whenever a match is found, the script most likely has behavioral detection present.

Category		Keyword
Event type	*Mouse*	`mousemove`
		`onmousemove`
		`mouseOverListener`
		`mousewheel`
		`DOMMouseScroll`
	Keyboard	`keydown`
Event–element connection		`trackEvent`
		`addEventListener`
		`addBehaviorkey`
Re-identification patterns		`distil`
		`perimeterx`
		`adscore`
		`datadome`
		`perfdrive`

Table 5.1: Behavioral detection categories

Detection method	Pattern	# Matches
Mouse movement	`mousemove`	4676
	`onmousemove`	1344
	`mouseOverListener`	126
Scrolling	`mousewheel`	1882
	`DOMMouseScroll`	1436
Keyboard	`keydown`	4726
Event trackers	`trackEvent`	2523
	`addEventListener`	6754
	`addBehaviorkey`	611

Table 5.2: Behavioral detection method occurrences

Besides the different patterns found above which can be classified under the same category as the detection patterns in table 4.1, there were also occurrences of new patterns for behavioral detection such as a code inclusion by Distil Networks. Distil Networks is known for its bot detection and its machine learning algorithms used for behavioral detection[1].

[1] https://www.imperva.com/

Whenever we take a closer look at the scripts we can see the methods work in practise. The most occurring methods 'mousemove', 'keydown' and 'mousewheel' are more often than not used to simply detect the movements of the user and adapt the buttons or content of the web page accordingly (cf. 5). The other methods which are less used, such as the 'mouseOverListener', are mostly used in the context of behavioral-based detection. Whenever we take a look at the full scripts using these code inclusions we can see that a full user report is built, based on the behavioral analysis.

As mentioned before, the most occurring code inclusions are more often than not used to detect movement of the user, unrelated to behavioral-based detection. Whenever these code inclusions are picked up however, they are treated as possible behavioral-based detection. In the grand scheme, this can lead to false positives by which effects the scoring mechanism with the relatively high amount of matches. For a more accurate representation of the numbers, an extra step to eliminate false positives should be taken. A next step towards reaching an even more accurate depiction of the amount of behavioral-based detection being done, is to use more specific regular expressions to eliminate the benign use of these code inclusions.

Chapter 6

Automatically recognising behavioral detection

We have chosen to research behavioral detection methods we have found both in the manual reverse analysis and in literature. Since we have not seen new methods in our research we can not be certain of the behavioral characteristics that are bound by the unknown behavioral detection methods. Therefore, at this stage we have deliberately chosen to not look for and investigate these unknown behavioral detection methods further, but instead focus on what we have found in the reverse manual analysis and the literature.

What can we detect and what do we choose for?

In this research we do not focus on finding behavioral detection patterns that we have not seen, in either literature or the manual reverse analysis. We have decided to shift our focus away from this, since we can not be certain of the characteristics of the behavioral detection pattern when we have not found it in the reverse manual analysis or in the literature. Therefore we are not sure of what we should look for and detection is therefore not feasible.

Table 4.1 gives the detection methods we have found in literature and table 5.1 shows the behavioral detection methods we have found in the manual reverse analysis and are used for automatically identifying behavioral detection.

Challenges and limitations

Automatic detection comes with its challenges. We are only able to perform client-side detection and no server-side detection. Besides that, when we do perform client-side detection, we can miss certain behavioral detection methods and misclassify others. We are able to miss certain behavioral detection methods since we can only identify detection methods we have encountered in the manual analysis or in literature. Besides that, we are able to misclassify certain methods as behavioral detection whenever we perform client-side detection.

Data acquisition

Acquiring data in this study means the collection of scripts containing behavioral detection methods. The most efficient way of acquiring this data is through an automated tool. This tool should be able to traverse the homepage of a web page and return the static scripts.

In this research area there are multiple tools that can be used [MSD+20]. There are HTTP clients such as wget, headless browsers such as PhantomJS and automated user browsers such as OpenWPM [Goß20].

At first, we tried using the automated user browser OpenWPM to measure the behavioral detection on the web. Regarding resource consumption, OpenWPM relatively uses a lot of resources compared to the other two classes of tools mentioned. However, the completeness of the OpenWPM tool with respect to scripts downloaded is a lot higher than the HTTP clients. The completeness is higher because the automated user browsers is dynamic whereas HTTP clients only find the static scripts. One major disadvantage the automated user browser OpenWPM has over the headless browsers and HTTP clients is the complexity of using the tool. Before using OpenWPM, the different commands and code snippets which can be found on the wiki of the tool need to be understood. Then, separate commands need to be programmed which are in line with the framework that is set up, to generate the results you are looking for. After running into issues locally and being able to fix them, these issues were unable to be resolved when running OpenWPM on a server. Websites which were reachable would give a 404 error (page not found) or a 403 error (forbidden) on multiple occasions. Since it would cost too much energy and time to fix this and results were more important, we decided not to use OpenWPM. OpenWPM is not trivially deployable.

We then decided to look into using headless browsers (e.g. PhantomJS) or a HTTP client (e.g. wget). Simplicity-wise, HTTP clients are easier in use. Using a headless browser for the first time requires some learning process, whereas using the HTTP client wget can be used immediately.

Moreover, the resource consumption of the HTTP client wget is lower than the headless browsers. Therefore we have opted to use the HTTP client wget for our research.

Design decisions

Scraping techniques and other bot-related activities on websites have evolved over the years. Many websites have implemented countermeasures against bots being able to circumvent their detection methods. One of the initial techniques to try and counter bots is the usage of obfuscation. Obfuscation is the task of deliberately changing the source code in such a way that it is difficult to read and understand. However, in the manual reverse analysis we have seen that obfuscation techniques do not effect the outcomes of our research as they do not effect the behavioral detection part of the scripts.

Another problem with the evaluation of the scripts is the fact that the vast majority of the scripts is minified. Minification can be seen as a type of obfuscation as some variables will be renamed and empty spaces are removed. The main purpose of minification however, is the reduction of the file size. The JavaScript beautifier atom-

beatify, which is a package within the atom IDE was used to make sure the scripts had their spacing restored.

After this process, we can start looking at the content of the scripts and investigate whether behavioral detection is being used. The scripts are statically analysed for behavioral detection patterns. The process of checking for behavioral detection patterns is done as described in section 6.2.

Moreover, the choice was made for static analysis instead of dynamic analysis. Whereas dynamic analysis has the advantage being certain that the code found is actually being executed and used, it has a major problem for this research. Namely, the code that is not being used at the time of analysis will not be picked up and analysed even though it is actually being used by the website. For this reason we have chosen for static analysis.

In the manual reverse-analysis we have found that some scripts are inconsistent with their usage of capital letters. Therefore we have made sure that every line in the script was converted to lowercase and matches the lowercase terms we used for analyzing whether the script contains behavioral-based detection methods.

```
1 line = f.readline().lower()
```

Code snippet 6.1: Lowercase conversion

Analyzing the files there were some problems at first. Due to obfuscations in encoding used by the scripts we needed to use codecs with the 'utf-8' encoding and we ignore the errors thrown. We ignore the errors instead of trying to solve them since it was found with manual analysis that these errors thrown were not of interest for the research (e.g. differently encoded search strings on the websites).

```
1 f = codecs.open(os.path.join(subdir, file), 'r',
2 encoding='utf-8', errors='ignore')
```

Code snippet 6.2: Using utf-8 encoding

Whenever analyzing the files, we came across another problem with the implementation. Whenever we would search for re-identification patterns we would generate a lot of false positives. For example, code inclusions such as .distilled would come up as being a part of the re-identification pattern Distil. Therefore, we needed to use regular expressions to limit the search for re-identification patterns to the exact term we gave the script instead of simply looping through the code inclusions we stamped as being behavioral-detection based and disregarding which extensions were added to the functions.

The re-identification pattern now looked like this:

```
1 if re.search(r"\b{}\b".format(detector),line):
```

Code snippet 6.3: Re-identification pattern regular expression

6.1 Recognising behavioral detection

To better recognise behavioral detection we have made use of both known methods found in literature and the manual reverse analysis. These results of these methods were then combined and used in an automatic tool, analysing the potential behavioral detection scripts from the top 10,000 Tranco list. We will find methods that we have encountered in either of these studies. However, other behavioral detection outside of the behavioral detection found in of the methods described above will not be picked up by our tool.

Creating our own analysis strategy, it's important to note that behavioral detection is done in a couple of ways as mentioned above and therefore it is important to have multiple detection categories. We have split up the detection in the following detection categories:

1. **Event types**
   In the manual analysis we have found multiple code inclusions in the form of event types, which have been traced back to behavioral detection (cf. 5). These code inclusion can indicate behavioral detection being in place, but can also lead to false positives.

2. **Event-element connections**
   Event-element connections are methods that attach events to specific elements. In the reverse-analysis we have seen that a particular subset of these events have been largely present in the behavioral detection scripts.

3. **Re-identification patterns**
   A re-identification pattern is a pattern which comes from a specific web-bot detector. For example, a company such as Distil Networks is known for its web-bot detection based on machine learning. The re-identification patterns are used for validating the identification of a behavioral detection script. Whenever a re-identification pattern is found, it is a certainty that behavioral detection is present.

6.2 A scoring mechanism for identifying behavioral detection

We re-use the concept of Vlot to determine a score based on the appearance of patterns (static check) [Vlo19]. However, the specific patterns used, the weights of the patterns and the aggregation are original.

To categorize the results we have made use of a scoring algorithm. A score is given to each of the scripts, based on the scoring mechanism we will explain below. If the accumulated score of a script goes above the threshold mentioned in this same section, the script will be labeled as a behavioral detection script.

After looking at the automated reverse-analysis and verifying the results with the manual reverse-analysis we came up with the following scoring mechanism. Instead of using arbitrary scores we have chosen to normalize the scores and use percentages to better depict the chance of behavioral detection being present. Based on the reverse

analysis we have determined a threshold which the combined detection categories need to exceed for the script to be classified as being a behavioral detection script.

Since the methods that attach events to specific elements have a high chance of being included when it comes to scripts not containing behavioral detection, the score of the event types is higher than the event-element connections (20%) at 50% for the mouse event types and 30% for the keyboard event types. The choices for these percentages are based on the amount of occurrences in the different scripts and the outcomes of score calculations after several test runs.

After several test runs and analysis of the results we have seen that the tool is accurate but still has some flaws. A scoring mechanism will always be flawed, but using the normalization we have made its accuracy more precise.

	Event Types	**Methods that attach events to specific elements**	**Re-identification patterns**
Code Inclusion	mousemove onmousemove mouseOverListener mousewheel DOMMouseScroll keydown	trackEvent addEventListener addBehaviorKey	distil perimeterx adscore datadome perfdrive
Threshold	Mouse 20 Keyboard 10	20	1
Score	Mouse +50% Keyboard +30%	+25%	100%

Table 6.1: Scoring mechanism

It is important to note that adding the score to the total score will only happen after the threshold is met or exceeded. These thresholds were determined based on the manual and automatic reverse analysis results.

Based on what we have seen in the reverse analysis we have decided to split up the scoring mechanism results into 5 separate categories. The first category consists of scripts which have no indications of behavioral detection. A score 0%-20% indicates the threshold was not met for any of the categories in table 6.1. The next category has some indications of behavioral detection in the form of keyboard event types. These event types, not in combination with other methods, are in our opinion not enough to classify a script as having behavioral detection. Category 3 contains the scripts we are uncertain about. These scripts have some indications of behavioral detection, but do not have any supporting methods. However, category 4 scripts do have multiple indications of behavioral detection being present. These scripts contain enough mouse event types to surpass the threshold and are accompanied by multiple event-element connections, keyboard event types or both. The category with the score of 100% represents the scripts of which we know for sure that they contain behavioral detection. These

scripts contain known patterns (re-identification patterns) from companies known to use behavioral detection. This categorical scoring system has been depicted below.

# indications detected	Estimation	Score
1. No indications	Not present	0%–20%
2. Low number of indications	Probably not	20%–40%
3. Mid number of indications	Uncertain	40%–60%
4. High number of indications	Likely	60%–95%
5. Re-identification pattern found	Certain	100%

Table 6.2: Estimation and scoring of behavioral detection

The files are analysed line by line to check for code inclusions of the terms in table 6.1. These code inclusions are checked for, by using regular expression. The following code snippets shows some examples of what the script was able to detect:

```
1 loadDataDomeScript(){
2     var e=document.createElement("script");e.type="text/javascriptcript"
3     }(window, document, "script", "https://js.datadome.co/tags.js"
4 }
```

Code Snippet 6.4: Detection pattern: Re-identification pattern (datadome)

```
1 function(){
2     h("DistilFP2Start");var t=new s; t.interrogate(function(t){h("
          DistilFP2End")
3 }
```

Code Snippet 6.5: Detection pattern: Re-identification pattern (distil)

```
1 verifiedCaptcha:!1,failedToVerifyCaptcha:!1,failedToDecodeBase64:!1,
      errorMessage:""}
2 ,n.env="public",n.serviceName="perimeterx"
```

Code Snippet 6.6: Detection pattern: Re-identification pattern (perimeterx)

```
1 document.addEventListener('touchstart', touchstartListener,
      eventListenersOptions)
2 document.addEventListener('mouseover', mouseoverListener,
      eventListenersOptions)
```

Code Snippet 6.7: Detection patterns: Mouse event (mouseoverListener) and Event-element connection (addEventListener)

Since we know from our research that the re-identification patterns such as the ones above all use behavioral detection these scripts will score 100%. However code snippet 6.7 shows 3 pattern matches. As we know, these can be false positives as the EventListeners and mouseOverListener can also be used for other purposes on the website. Consequently, we would need to analyze the entire script to see whether these categories surpass the threshold.

Chapter 7

Identifying behavioral detection in the wild

This chapter serves to explain the methodology behind the case study as proposed by us, the researchers. The methodology is created to investigate behavioral detection methods on the internet and its existing countermeasures. The methodology has been inspired by the methodology from the thesis of Gabry Vlot [Vlo19]. Similar to the thesis of Gabry Vlot we will start with a gathering phase where the scripts will be collected for manual reverse analysis. Subsequently, the manual analysis will take place, a literature study will be done and thereafter the reverse analysis will be automated. Afterwards, behavioral detection will be measured on the web using the insights we have gained from the reverse analysis and the literature study. This poses the question whether we find behavioral detection by scripts we have not encountered before. If so, do these scripts indeed detect bots by their behavior?

7.1 Methodology

We visit the homepages of the top 10,000 websites from the Tranco top 1 million. The running scripts are extracted and downloaded. The extraction process was done using wget and is explained in the data acquisition paragraph of chapter 6. After the data acquisition the scripts are checked for behavioral detection.

7.1.1 Setup

Initially, we have split up the detection methods, where it is important to note that its scoring mechanism is prone to false positives as mentioned in Section 7.1.3.

Pattern recognition is a common technique to figure out whether someone or something is out of place. For example, in airports, screening supervisors will have score sheets and use security camera footage to evaluate whether someone's behavior is deemed as suspicious behavior.[1] This course of action has now been transferred over into bot detection, which is where the name behavioral detection comes from. Similar to the screening of people in airports, our bot detection scripts have a threshold and whenever this threshold is exceeded, the behavior of the bot is classified as suspicious. From the results of the case study we can conclude that the patterns mentioned in Section 6.1 are sophisticated enough to check whether scripts use behavioral-based detection.

We have decided to split the behavioral detection methods up into categories. All three of these categories have been established through results from the manual reverse analysis and the literature study. One of these categories is event types and contains Java event types that have been linked to behavioral detection in either the literature study or in the reverse analysis.

Besides that, these methods often have other methods which attach these events to specific elements. These methods that attach events to specific elements have a separate category which can further confirm the suspicions created by the first category.

Furthermore, re-identification patterns are used to confirm the suspicions of behavioral detection being present. Through reverse-analysis we found that re-identification patterns are the best indication of behavioral detection being present. Even though re-identification patterns are still able to be false positives (e.g. inclusions of these re-identification patterns in comments), it is more than likely that behavioral detection is being used whenever these re-identification patterns are included. Moreover, it is also possible that these re-identification patterns are found when no other methods relating to behavioral detection have been found. In this case, behavioral detection is

[1] https://www.asi-mag.com/behaviour-detection-demystified-glimpse-tsas-capability/

most likely done server-side and our tool is unable to analyze the code for behavioral detection event types. Therefore we have decided that these re-identification patterns always return a 100% score, meaning the script has behavioral detection whenever one of these patterns is encountered.

Running a script locally

To better understand the workings of behavioral detection scripts, it is necessary to study their behavior. Running one of these scripts locally is the first step into understanding its behavior regarding the recording and storing of data. However, studying these scripts more carefully suggests that almost all scripts make use of API calls. These get in the way of running these scripts locally. Therefore, we have set up a proxy which negates the problem of getting timeouts.

Besides having to set up a proxy, some scripts are scrambled to make sure they are as unrecognizable as possible. We take a look at one of the many scripts which is labeled "Tocca.min00.js". The min00 suggests we are dealing with minified javascript. Using a JavaScript beautifier we are able to undo the minification and make the JavaScript readable. Now, trying to run the script gives us multiple errors. Syntax errors, variable definition errors and callback errors. For this particular script we were able to keep the functionality of the script intact whilst at the same time being able to both fix all of the errors above as well as generate output.

For example, we decided to take a look at the mouse movements and mouse presses. Whenever the mouse button is pressed, we would see the following alert:

Figure 7.1: Mouse click alert

Both pointerdown and mousedown indicate that the mouse button was pressed and a 'touchstart' event is started to track what happens right after the mouse button is pressed. After pressing the mouse button we moved the mouse whilst still holding the left mouse button.

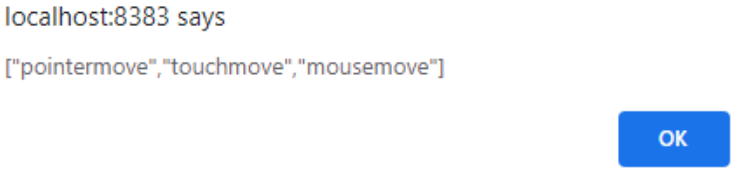


Figure 7.2: Mouse dragging alert

As we can see in the figure above, the pointer and the mouse are being moved, but the event is called 'touchmove', meaning we are still holding the left mouse button. In short, the 'touchstart' event has recorded the fact that we are still holding the left mouse button and dragging the mouse across the screen. However, when it comes to behavioral detection this information is useless by itself. To create a full user profile and to check whether any inhuman movements are done with the mouse, we need both timestamps and pointer coordinates. Respectively, this is what is being recorded. Further results regarding the local script analysis can be found in appendix C.

Whilst running a script locally it became clear how the recording and storing of attributes was done. Using this information in combination with the reverse analysis and the literature study we were able to come to the conclusion that full user profiles are built. This data is stored on a web server and the website owner or a third party is able to look at the user profiles and possibly contains an AI generated threat level. As we were able to see in figure C.1, timestamps are used also. We were able to find that these timestamps are not only used to check which action was taken by the user at which time, but that this plays a big part in behavioral-detection as well. Namely, time tamps are used for rate limiting and analyzing how often and how rapid requests are made to the server. All of the alerts shown in section 7.1.1, will be combined to form a user profile and possibly a threat level. Whenever the threat level is above a certain threshold, these user agents can be flagged for closer monitoring, given a CAPTCHA to solve to make sure they are in fact not bots or they can be denied access entirely by placing the user agent on a blacklist.

7.1.2 Results

Whenever measuring the degree of behavioral detection on a website we take a look at the code inclusions of the different patterns mentioned in section 5.

Detection method		**Found method**				**Example**
		1	2	3	4	
Event type	*Mouse*	x	x		x	www.verizonmedia.com— 2.6e5a4b21.chunk.js
			x		x	www.mobile.de — app.js
		x		x	x	www.kapwing.com— ffmpeg-core.js
		x	x			www.flashscore.com— core_2_1841000000.js
	Keyboard	x		x	x	www.telstra.com.au — latest.min.js
		x		x	x	www.umbc.edu — swiper.js
		x		x	x	www.microchip.com — kendo.min.js
		x		x	x	www.jnu.edu.cn — iscroll.js
		x		x	x	www.pepperdine.edu — main.bundle.js
		5	6	7		
Event-element connection		x				www.redlink.com.ar — raptor.min.js
		x	x			www.flinders.edu.au — at.js
		x		x		www.astm.org — addtl.js
		x	x			www.www.mturk.com — s_code.js
		8	9			
Re-identification patterns		x				www.whitepages.com — dstl-wp.js
		x				www.www.streetinsider.com — dstlstrtins.js
			x			www.walmart.com — main.4fb314cb.chunk.js
			x			www.skyscanner.com — main.66eba5ab.js
		x				www.oprah.com — dstl-oprh.js

Methods:

1. mousemove/onmousemove
2. mouseOverListener
3. mousewheel/DOMMouseScroll
4. keydown
5. addEventListener
6. trackEvent
7. addBehaviorKey
8. distil
9. perimeterx

Table 7.1: Several sites per behavioral detection method

In table 7.1 we see that scripts contain certain methods that are classified by us as scripts potentially containing behavioral detection methods. However, these scripts first need to exceed a certain threshold to be classified as being a behavioral detection script.

In the table below we can see how many occurrences of each of the methods is necessary to be recognised as possible behavioral detection on its own. Mouse movement and keyboard event types have a lower threshold than event-element connections since the event-element connections on their own are no indication of behavioral detection. Once they are used in combination with the other methods it becomes a vital part of

information processing in the behavioral detection. The re-identification patterns do not have a threshold to surpass since the presence of this pattern means that behavioral detection is present.

Detection method		Threshold
Event type	*Mouse*	10
	Keyboard	10
Event-element connection		20
Re-identification patterns		0

Table 7.2: Amount of websites per detection method

Threshold	Scripts in range range (amount)	Scripts in range (%)
0-20%	17203	97.29
20-40%	290	1.62
40-60%	97	0.54
60-95%	69	0.39
100%	23	0.13

Table 7.3: Measurement of behavioral-based detection

The table above depicts that for most of the scripts we can conclude for certain that no behavioral detection is present. For only 23 of 17,862 scripts we can conclude that we are in fact 100% certain they carry out behavioral detection.

7.1.3 Validation

The validation of the findings is important to justify the outcomes presented. To justify the outcome of the wget measurement and our accompanying conclusion it is important to manually look at some of the scripts in question and check whether the tool did not return a false positive.

False positives in this case means that there are scripts labeled as behavioral detection, which do carry out behavioral detection. The way to validate these scripts and reduce the false positive rate is by manual validation and random sampling.

False negatives in our research are the scripts which are not labeled as behavioral detection scripts, but in reality do carry out behavioral detection. Besides that, the behavioral-based detection script was analyzed for correctness. This correctness was both validated by manual reverse-analysis to confirm the results as well as change variables and compare the results to the expected output.

In the first phase of the research we have taken a look at scripts previously found in a bot detection research [JKV19]. Reverse-analyzing these files and using its data

means that we can be certain that the input we use for the script is bot-detection and has a chance of being behavioral-based detection.

The fact that we cannot be certain whether behavioral-based detection is present, comes from the fact that detecting whether detection is done is simply hard to verify. For example, mouse movement can be used to track and sketch a data profile for examination regarding behavioral-based detection. On the other hand, these mouse movements might as well be used for another purpose, such as checking the most popular areas of a website.

This problem can lead to false positives, by falsely identifying benign usage of these code inclusions as use for behavioral-based detection. For this reason, percentages were used to have a chance-based scoring system regarding the scripts of the websites we have analyzed. The higher the percentage, the more likely behavioral-based detection is present . Other measures have been taken to try and minimize the false positives such as using regular expressions to minimize the false identification of re-identification patterns as well as using reverse-analysis on the results to remove certain patterns we have used in past iterations.

False positives are the main issue that we came across. However, we still have to keep in mind that false negatives are a possibility. Missing certain behavioral-based detection which is present, but is not picked up by our script. To mitigate this issue, we have done multiple sample tests of the results and manually analyzed these scripts for behavioral-based detection. Obviously, we are unable to analyze all of these scripts by hand and automated scripts might still miss some behavioral-based detection. However, since we do not have the resources to manually analyze all of these scripts, a manual reverse-analysis of the scripts analyzed by the script was our best alternative.

Category	Population	Proportion	Sample size	FP	FP%
0-20%	17203	97.29	41	2	5%
20-40%	290	1.62	23	2	9%
40-60%	97	0.54	8	0	0%
60-95%	69	0.39	6	0	0%
100%	23	0.13	2	0	0%

Table 7.4: Amount of false positives

The amount of false positives was calculated through random sampling over the population of 17,862 scripts. Meaning to calculate the necessary sample size for a statistically significant result for simple random sampling we have to take account the proportion as well. The minimum number of necessary samples to meet the desired statistical constraints were calculated using scientific a scientific calculator as well as an online tool. On the website of this same tool is explained how this calculation works in practise[2].

After calculating the minimum sample size requirements with a confidence level of

[2]https://www.abs.gov.au/websitedbs/d3310114.nsf/home/sample+size+calculator

95%, the corresponding amount of scripts were randomly selected from the different categories. These scripts were then manually reverse analyzed with the conclusion whether they should be in their current category. We found that the scripts ranging from 40% up until 100% were all in the correct category. However, we found that 2 scripts in the 20%–40% category should be in the 40%–60% category as it is very unclear whether behavioral detection is present. Moreover, 2 scripts from the first category (0%–20%) can actually be placed in the second category (20%–40%) since there are some very limited behavioral detection methods and we still do not believe behavioral detection is present.

Chapter 8

Conclusions and Future work

The aim of this research is to investigate how behavioral detection works in practise. Besides that, the key subgoal is to uncover when current real-world bot detection methods deem evidence sufficient to classify a visitor as a bot.

8.1 Conclusions

By conducting the research explained in the methodology we have answered the research questions below:

Sub-question 1: Which methods for behavioral detection exist?
We started with identifying which behavioral detection methods were discussed in literature. The results are depicted in the table below:

Detection method	**Countermeasure**	**Discussed by**	**Detected at**
Rate limiting of user requests	Delays and random waits	[BMV20, MD11]	client or server
Honeypot trap	Do not access invisible elements	[SA18]	client and server
Captcha/Recaptcha	– Audio-version → speech-to-text – Crowdsourcing	[SA18, SKSP17, BMV20]	client and server
Site traversal	Human-like path traversal configuration	a.o. [TCK09, SVA11]	server
Mouse movement	Random mouse movements	[PB04, CGK+12]	client and server

Table 8.1: Overview of behavioral detection methods discussed in literature

Subsequently, we performed a manual reverse analysis of the behavioral-based detection scripts for other existing behavioral detection methods. We were able to categorize the behavioral detection methods in the categories: Event type with sub-sections Mouse and Keyboard, Event-element connection and Re-identification patterns as can be seen in the table below:

Category		Method
Event type	*Mouse*	`mousemove`
		`onmousemove`
		`mouseOverListener`
		`mousewheel`
		`DOMMouseScroll`
	Keyboard	`keydown`
Event–element connection		`trackEvent`
		`addEventListener`
		`addBehaviorkey`
Re-identification patterns		`distil`
		`perimeterx`
		`adscore`
		`datadome`
		`perfdrive`

Table 8.2: Behavioral detection categories

Sub-question 2: How can behavioral detection be recognised?
Up until now, a strategy to detect behavioral-based scripts has not been publicly published. In [Vlo19] Vlot has used a method for detecting scripts which detects bot detection scripts. This method was used to get an understanding of how to go about detecting bot-detection scripts. Besides that, the scripts we used were specifically created to analyze the scripts for behavioral-based detection.

Creating our own strategy, it's important to note that behavioral detection is done in a couple of ways and therefore it is important to have multiple detection categories. We have split up the detection in the following detection categories:

1. **Event types**
   In the manual analysis we have found multiple code inclusions in the form of event types, which have been traced back to behavioral detection (cf. 5). These code inclusion can indicate behavioral detection being in place, but can also lead to false positives.

2. **Methods that attach events to specific elements**
   Methods that attach events to specific elements are indirectly linked to behavioral detection. In the reverse-analysis we have seen that a particular subset of these events have been largely present in the behavioral detection scripts.

3. **Re-identification patterns**
   A re-identification pattern is a pattern which comes from a specific web-bot detector. For example, a company such as Distil Networks is known for its web-bot detection based on machine learning. The re-identification patterns are used

for validating the identification of a behavioral detection script. Whenever a re-identification pattern is found, it is a certainty that behavioral detection is present.

One of the biggest challenges with identifying behavioral detection is trying to distinguish between programs that only use behavioral detection features and scripts that are created with the intent of behavioral detection. For example, drawing programs need to track mouse movement and uses its corresponding methods which attach events to specific elements. However, in our manual analysis of the scripts and drawing programs such as `www.draw.io` we found out that most of these drawing programs use multiple JavaScript files to track movement of the objects, the mouse or keyboard presses. From the manual reverse analysis we were able to conclude that in behavioral detection scripts, these methods were always present in relatively large amounts. Therefore, we are often able to distinguish the benign use of behavioral detection features in scripts from the scripts which are created with the intent of behavioral detection by the amount of behavioral detection features the script contains.

To classify the scripts as behavioral detection scripts we need to make use of a scoring-algorithm. Even though scoring mechanisms are flawed there is no perfect way to determine whether detection is being done. We are able to detect whether the mouse movement is detected and stored, but whether the mouse movements are then evaluated with behavioral-based detection is hard to detect. Therefore there are no obvious yes or no answers when it comes to checking whether a script uses behavioral-based detection. Instead of using arbitrary numbers, we normalize the results and have done multiple test runs adapting the scoring mechanism so that the scripts are now analyzed and a chance of behavioral-based detection is calculated and checked against a threshold. This has led us to the following scoring categories:

Estimation	**# indications detected**	**Score**
1. Not present	no indications	0%–20%
2. Probably not	low number of indications	20%–40%
3. Uncertain	mid number of indications	40%–60%
4. Likely	high number of indications	60%–95%
5. Certain	re-identification pattern found	100%

Table 8.3: Estimation and scoring of behavioral detection

Sub-question 3: How many sites are using behavioral detection?

Category	Population	Proportion	Sample size	FP	FP%
0-20%	17203	97.29	41	2	5%
20-40%	290	1.62	23	2	9%
40-60%	97	0.54	8	0	0%
60-95%	69	0.39	6	0	0%
100%	23	0.13	2	0	0%

Table 8.4: Amount of false positives

We can conclude from this table that 17,203 out of the 17.862 scripts have between 0% and 20% chance of containing behavioral-based detection.

Thus, we can conclude with certainty that 17,203 scripts do not use behavioral-based detection if we disregard any false negatives. 290 scripts have between 20% and 40% chance of containing behavioral detection. Meaning, 290 scripts possibly have some small form of behavioral-based detection, but have such a low chance that we believe no behavioral detection will be present. 97 scripts are in a range where we simply do not know whether they carry out behavioral detection until closer inspection as they do contain a limited amount of behavioral detection methods. However, 69 scripts have a good chance of containing behavioral-based detection since they contain multiple behavioral detection methods and the 23 scripts with the highest chance of containing behavioral detection certainly do contain behavioral detection.

There is a very limited amount of false positives in the population which was concluded through simple random sampling. However, the sample size taken was the minimum and using a larger sample size might alter the results slightly. Since of the limited time to conduct this research we were unable to use bigger sample sizes, but wanted to make sure that the results were statistically significant. We used a confidence level of 95%.

8.2 Future work

As expected, the research area we explore with this research is a largely unexplored information-rich area. We have adapted Vlot's score-based approach for detecting fingerprint bot detectors to detect incidents of behavioral detection on the web. As we have mentioned before, any score-based approach is flawed. Therefore further research could be done on creating another system that is fit to analyse scripts for behavioral-based detection or bot-detection in general.

Even though we have explored and described countermeasures against the behavioral detection methods we found, there will be more development regarding these countermeasures. It will be interesting to see whether more countermeasures will be developed which have a big impact on this research-area.

Moreover, the script we use to analyze the potential behavioral detection scripts can be improved in accuracy. An even bigger pool of behavioral detection methods and patterns can lead to more accuracy of the tool. As mentioned by Vlot in [Vlo19] introducing anti-patterns can also improve the accuracy of the tool in question.

We make use of static analysis using wget. Results using dynamic analysis with a more sophisticated scraping tool will possibly yield more significant results. Besides that, this research has laid the foundation of ingredients necessary for machine learning on scripts to detect behavioral detection in scripts and classify them accordingly. Further research using these more advanced techniques is the appropriate and logical next step to take regarding research in this area.

Since we have conducted this research on the homepages of the top 10,000 websites in the Tranco top 1 Million there is still a lot of room for further research sample-wise. It will be interesting to see whether a larger sample pool will alter the results. The top 10,000 websites contain many bigger websites which might contain more or less behavioral-based detection than the websites further down the list.

Bibliography

[BMV20] Aanshi Bhardwaj, Veenu Mangat, and Renu Vig. Effective mitigation against iots using super materials for distributed denial of service attacks in cloud computing. *Materials Today: Proceedings*, 28, 2020.

[BXZH14] Quan Bai, Gang Xiong, Yong Zhao, and Longtao He. Analysis and detection of bogus behavior in web crawler measurement. *Procedia Computer Science*, 31:1084–1091, 2014.

[CGK+12] Zi Chu, Steven Gianvechhio, Aaron Koehl, Haining Wang, and Sushil Jajodia. Blog or block: Detecting blog bots through behavioral biometrics. In *Computer Networks*, volume 57, pages 634–646. ScienceDirect, 2012.

[Dol19] Rick Dolfing. Towards a fingerprint surface. Master's thesis, Open Universiteit Nederland, 2019.

[EN16] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proc. 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pages 1388–1401. ACM, 2016.

[Goß20] Daniel Goßen. Design and implementation of a stealthy openwpm web scraper. Master's thesis, Radboud Universiteit Nijmegen, 2020.

[HE17] Rabih Haidar and Shady Elbassuoni. Website navigation behavior analysis for bot detection. In *Proc. International Conference on Data Science and Advanced Analytics (DSAA)*, pages 60–68. IEEE, 2017.

[IKT+19] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilios Katos, Stefanos Vrochidis, and Ioannis Kompatsiaris. Towards a framework for detecting advanced web bots. In *Proc. 14th International Conference on Availability, Reliability and Security (ARES'19)*, pages 1–10, 2019.

[JHZ+14] Yuede Ji, Yukun He, Dewei Zhu, Qiang Li, and Dong Guo. A mulitiprocess mechanism of evading behavior-based bot detection approaches. In *Information Security Practice and Experience - 10th International Conference, ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings*, volume 8434 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2014.

[JKV19] Hugo Jonker, Benjamin Krumnow, and Gabry Vlot. Fingerprint surface-based detection of web bot detectors. In *Proc. 24th European Symposium*

on Research in Computer Security Part II (ESORICS'19), volume 11736 of *LNCS*, pages 586–605. Springer, 2019.

[KWS+10] Hongwen Kang, Kuansan Wang, David Soukal, Fritz Behr, and Zijian Zheng. Large-scale bot detection for search engines. In *Proc. 19th International Conference on World Wide Web (WWW '10)*, pages 501–510. ACM, 2010.

[MD11] Steve Mansfield-Devine. Ddos: Threats and mitigation. *Network Security*, 2011:5–12, 2011.

[MSD+20] Kunal Mehta, Maya Salvi, Rumil Dand, Vineet Makharia, and Prachi Natu. A comparative study of various approaches to adaptive web scraping. In *ICDSMLA 2019*, pages 1245–1256, 2020.

[PB04] Maja Pusara and Carla Elizabeth Brodley. User re-authentication via mouse movements. In *Proc. of the 2004 ACM workshop on Visualization and data mining for computer security (VizSEC/DMSEC'04)*, pages 1–8. ACM, 2004.

[SA18] Boukari Souley and Huawa Abubakar. A captcha-based intrusion detection model. *International Journal of Software Engineering & Applications (IJSEA)*, 9:29–40, 2018.

[SKSP17] Saumya Solanki, Gautam Krishnan, Varshini Sampath, and Jason Polakis. In (cyber)space bots can hear you speak: Breaking audio captchas using ots speech recognition. In *Proc. 10th ACM Workshop on Artificial Intelligence and Security (AISec'17)*, pages 69–80. ACM, 2017.

[SLG19] Michael Schwarz, Florian Lackner, and Daniel Gruss. Javascript template attacks: Automatically inferring host information for targeted exploits. In *NDSS'19*, 2019.

[SVA11] Dusan Stevanovic, Natalija Vlajic, and Aijun An. Unsupervised clustering of web sessions to detect malicious and non-malicious website users. *Procedia CS*, 5:123–131, 2011.

[TCK09] I-Hsien Ting, Lillian Clark, and Chris Kimble. Identifying web navigation behaviour and patterns automatically from clickstream data. *International Journal of Web Engineering and Technology*, 5:398–426, 2009.

[TK02] Pang Ning Tan and Vipin Kumar. Discovery of web robot sessions based on their navigational patterns. *Data Mining and Knowledge Discovery*, 6:9–35, 2002.

[Vlo19] Gabry Vlot. Automated data extraction: what you see might not be what you get. Master's thesis, Open Universiteit Nederland, 2019.

Appendix A

Implementation details

A.1 Implementation of wget

We design and adapt a way to measure incidents of behavioral detection on the web. Ultimately, the choice was made to use wget.

We want to make sure we only accept JavaScript as these scripts are written in JavaScript. To do this, wget has the flag -A which is then followed by js to ensure we only accept JavaScript.

```
wget -A js
```

The next step is to make sure that wget is invoked with the top 10,000 websites in the tranco top 1 million[1]. To do this we first create a .txt file containing the top 10,000 websites of the tranco 1 million. Then, using the -i flag in wget we can specify this .txt file as the input file and it will iterate over the websites.

```
wget -A js -i websites.txt
```

By running multiple tests it became clear that trying to connect to a website multiple times was never successful and takes up a lot of time. Therefore the choice was made to only give wget 1 try per IP listed for the website at hand. Besides that, it also became clear that the download was blocked on some occasions even though we would only visit the homepage. Therefore we turned off robots.txt, which needs the -e flag to be turned off. Moreover, it became clear the some files would be downloaded multiple times, meaning it would take up resources unnecessarily. Therefore we added the flag –no-clobber which makes sure that whenever a file is downloaded in the same directory it will be clobbered or overwritten upon repeated download. It is important to realise that we create a directory for each web page visited and therefore do not delete the same files if they are ran on different web pages.

```
wget --no-clobber --tries=1 -e robots=off -A js
-i websites.txt
```

As mentioned before we are only visiting the homepage. To ensure this, we disallow retrieval of links of the hierarchy above the directory of the homepage. Which means

[1] https://tranco-list.eu/

the ascent to a parent directory is not allowed whenever possible. This is done by using the -np parameter. In combination with the –page-requisites parameter, wget is prompted to download all the files which are needed to display the web page.

```
wget --no-clobber --tries=1 -e robots=off -np
--page-requisites -A js -i websites.txt
```

To make sure we do not get blocked from visiting websites we change our user agent to a common chrome user agent. Furthermore, recursive download can overload remote servers which is why we don't use recursive downloading. Otherwise, administrators might ban a user from accessing the website. using –random-wait we delay the downloading of files to not alarm admins.

```
wget --user-agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebkit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36" --random-wait --no-clobber --tries=1
-e robots=off -np --page-requisites -A js -i websites.txt
```

Wget was ran on a 1 Intel Xeon CPU Virtual Private Server with 100GB storage and 1GB memory. The operating system of the server was Ubuntu 20.04.

To connect to the Virtual Private Server we used ssh and we connected using the command:

```
ssh username@server_ip
```

After running wget and having the results stored on our Virtual Private Server we transferred the files to our local machines for manual analysis.

```
sudo scp -r username@server_ip:/home/username
/home/local/Documents/Thesis
```

A.2 Behavioral detection script

This script was created with the intent to iterate over each of the files downloaded using the tool mentioned above. In this section we will refer to this script as the Iterator file. The implementation of this script is explained in A.2.3.

The script is created to do the following:

1. create a log file for all patterns found (section A.2.1)
2. create files with the different the scripts that surpass the different thresholds (section A.2.2)

A.2.1 Log file

The log file created shows all different behavioral detection patterns found and in which file they are found. This file is created with the intent of validation as well as to gain more insight in the amount of occurrences for each of the patterns.

A.2.2 Threshold files

The threshold files show us the files which surpass the threshold as mentioned in section 6.2. Every line in the file starts with a percentage and is followed by the file belonging to the chance of behavioral detection being present.

A.2.3 Iterator file

The first step in the implementation was to specify the directory which contains the files which need to be analysed. In our case this is the rootdir file, where client_21042_1 is the name given to us by the server hosting company. Afterwards, we create the three files mentioned above to store our results. The outfile in this case is the log file mentioned above.

```
rootdir = '/home/thatmitch/Documents/scrape/client_21042_1'
outfile = '/home/thatmitch/Documents/scrape/output.txt'
```

To store the amount of occurrences for each of the different patterns we then created the variables 'id_var', 'mouseeventtype_var', 'keyboardeventtype_var' and 'eventelement_var'. To make sure we know how far our analyzer is into the process we also use the variable 'analyzer' and to calculate the chance of behavioral detection being present we use the variable 'chance'. Now we can start iterating over all of the files in the rootdir. For each of the files we increase our analyzer variable to see the progress. Analyzing the files resulted in encountering some problems at first. Due to obfuscations in encoding used by the scripts we needed to use codecs with the 'utf-8' encoding and we ignore the errors thrown. We ignore the errors instead of trying to solve them since it was found with manual analysis that these errors thrown were not influential on the results of the research (e.g. differently encoded search strings on the websites). We have opted to use a [try, except, finally] block to make sure the script has no problems reading the files and otherwise throws an error. To mitigate problems with the files we use the [finally] block to close the file after it is analysed.

Each file is analysed line by line using the function

```
.readline()
```

It was also found in the manual analysis that websites have different capitalisation for the bot detection patterns and therefore we have opted to make everything lowercase by using the function

```
.lower()
```

This leads to the following framework:

```
for root, dirs, files in os.walk(rootdir):
    for file in files:
        analyzer += 1
        f = codecs.open(os.path.join(subdir, file), 'r',
        encoding='utf-8', errors='ignore')

        try:
            line = f.readline().lower()
            while line:
                ...
                line = f.readline().lower()

        finally:
            f.close()
```

Code Snippet A.1: Framework

For each line we iterate, we check for the different patterns. Whenever one of these patterns is detected, the variable corresponding to this pattern is increased by 1. Besides that, the log file will now contain a line which states which pattern was found in the corresponding file with its filename following the pattern. This code is located on the dots of the section of code above.

```
for idpattern in idpatterns:
    if re.search(r"\b{}\b".format(idpattern),line):
        id_var += 100
        out.write(idpattern + " found in " + os.path.join(root, file)

for mouseeventtype in mouseeventtypes:
    if mouseeventtype in line:
        mouseeventtype_var += 1
        out.write(mouseeventtype + " found in " + os.path.join(root, file
            )

for keyboardeventtype in keyboardeventtypes:
    if keyboardeventtype in line:
        keyboardeventtype_var += 1
        out.write(keyboardeventtype + " found in " + os.path.join(root,
            file)

for eventelement in eventelements:
    if eventelement in line:
        eventelement_var += 1
        out.write(eventelement + " found in " + os.path.join(root, file)
```

Code Snippet A.2: Pattern detection

We then write the scores to the different threshold files. Afterwards, the variables in the form of integers are now checked for occurrences. For each occurrence of the re-identification pattern (id_var > 0), the chance of the file containing behavioral detection becomes 100%. The other patterns each have their own percentage based on the reverse analysis. Then, the total chance of behavioral detection being present is checked against a threshold and whenever this threshold is met or exceeded, the filename will be written to the threshold file with its corresponding chance. Then, all variables are set to 0 and the process is repeated for the other files.

```
1
2  if 0 <= endresult < 20:
3      website0 = True
4  if 20 <= endresult < 40:
5      website20 = True
6
7  ...
8
9  for dir in dirs:
10     if website0:
11         if os.path.join(root,dir) not in websitelist0:
12             res = os.path.join(root,dir).partition('.com')[0]
13             websitelist.append(res)
14
15     ...
```

Code Snippet A.3: Behavioral detection chance calculation

Appendix B

Methods and countermeasures discussed in literature

This appendix serves to give a more detailed explanation and code examples regarding the behavioral detection methods and their countermeasures discussed in literature. The detection method is followed by a code example, showing the detection method in practise. Then, the papers in which the detection method is discussed are shown. Below that, a description of the detection method is given, explaining how the detection method is used for behavioral detection. Lastly, the countermeasures are briefly mentioned and then their implementation and usage are discussed in detail.

B.0.1 Mouse movement

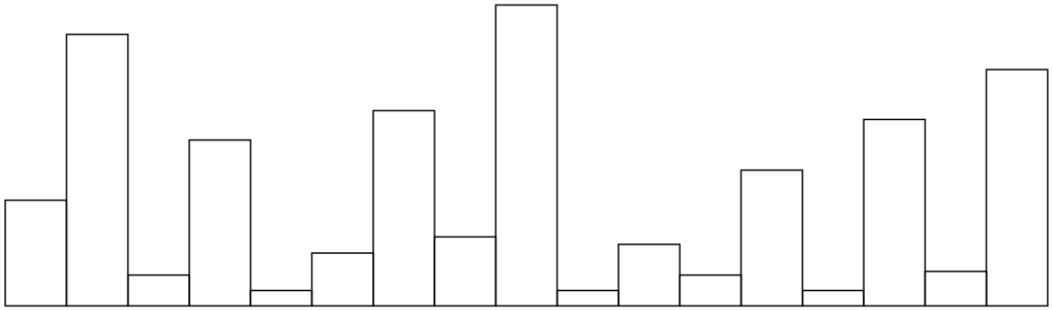
```
1 function mouse_position(){
2     var e = window.event;
3     var posX = e.clientX;
4     var posY = e.clientY;
5     document.Form1.posx.value = posX;
6     document.Form1.posy.value = posY;
7     var t = setTimeout(mouse_position,100);
8 }
```

Code Snippet B.1: Mouse movement

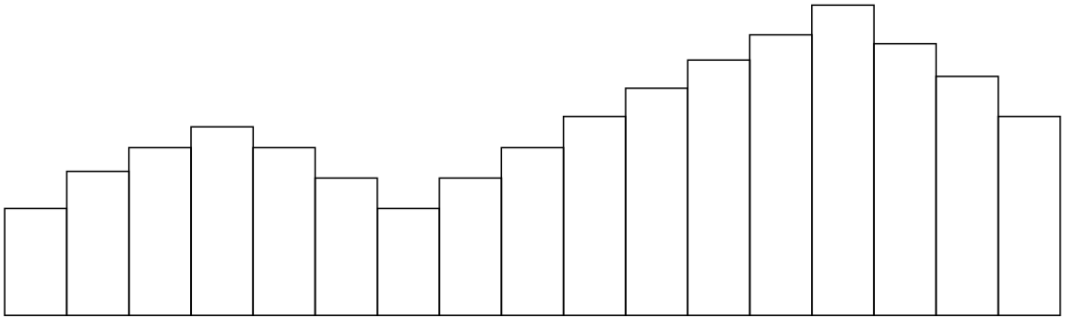
Discussed by (a.o.): [PB04] and [CGK+12].

Description: Keeping track of mouse movement is a simple trick to keep track of the users' interaction with the mouse. If the mouse moves linearly, does not move at all, or movements which conform to a pattern, this will be picked up by the behavioral detection tool.

Countermeasure: Random mouse movements (e.g. the simplex noise algorithm). Whereas completely random algorithms will give this result:



The random algorithm in combination with the simplex noise algorithm will give us:



The random algorithm only looks at one of the parameters. In this case, it only looks at the x-axis and will completely randomize the y-axis. However, we need to make sure that both the x-axis and y-axis are taken into account to make sure the mouse does not move linearly, nor erratically. This means the randomness of the algorithm is kept, but instead it is limited to within certain boundaries. This is to smoothen out the curve as can be seen in the second image. The noise script is loaded and used on every single frame to simulate the mouse movement. An important feature of this countermeasure is the fact that both the randomness and speed can be adjusted. Therefore it will be hard for machine learning to pick up on mouse patterns.

B.0.2 CAPTCHA/ReCAPTCHA

```
1    <script src='https://www.google.com/recaptcha/api.js'></script>
2    <div class="g-recaptcha" data-sitekey="Your site key goes here"></div
         >
```

Code Snippet B.2: Recaptcha

Discussed by (a.o.): [SA18], [KWS+10], [SKSP17] and [BMV20].

Description: reCAPTCHA and CAPTCHA work on the same principles. They both use images which then need to be either written out (whenever the reCAPTCHA image is a word) or they need to be selected according to a certain category (e.g. select all cars or select all traffic lights). Whenever the word is typed correctly or the images selected are in line with the solution, the system acknowledges you are a human user and you are able to continue traversing the website. However when you are unable to solve the CAPTCHA, this will lead to the CAPTCHA being retriggered until you

have convinced the system you are indeed a human user. Otherwise content will be blocked.

Countermeasure: Since CAPTCHA and reCAPTCHA have been around for quite some time now, people have started developing countermeasures against this anti-bot measure. Crowdsourcing can be used to solve the CAPTCHA's for the bot. By outsourcing the process of solving the CAPTCHA to a distributed human workforce who can solve this CAPTCHA virtually, a bot will be able to circumvent the verification step.

For example, 2Captcha and Puppeteer can be used to automatically try and solve the CAPTCHA's being thrown[1]. Working with two API endpoints, where the first request made, retrieves the data which is needed to solve the CAPTCHA and then gives us the request ID. Whenever we come across a CAPTCHA which contains an image, the image will be converted to base64. Then, requests will be made to the API endpoint, polling until a solution is ready to be used. Visually, this works as follows:

Figure B.1: 2CAPTCHA request

Figure B.2: 2CAPTCHA response

[1]https://medium.com/@jsoverson/bypassing-captchas-with-headless-chrome-93f294518337

With reCAPTCHA this works the same, except for the fact that you need the sitekey instead of the request ID. Conveniently, this can be found on the of the reCAPTCHA, even if the iframe has not loaded.

Besides this form of countering the reCAPTCHA, unCAPTCHA is a service to solve reCAPTCHA, which has been around for quite some time now[2]. unCAPTCHA works on the fact that reCAPTCHA has an audio-mode, created for visually impaired people. The audio is downloaded by unCAPTCHA, dissected into multiple audio segments which are all uploaded to multiple speech-to-text services. Using a weighted vote, the reCAPTCHA is then filled in and solved with currently over 90% accuracy.

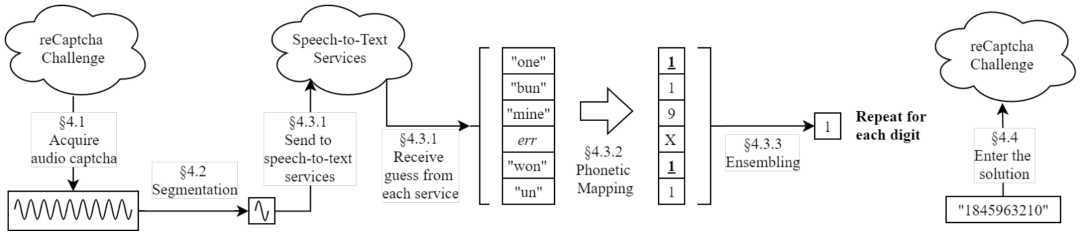


Figure B.3: unCAPTCHA process

[2]http://uncaptcha.cs.umd.edu/

B.0.3 Rate limiting of user requests

```
import * as db from "./db";

function update({ tokenCount, timestamp }, { interval, bucketCapacity },
    now) {
  const increase = Math.floor((now - timestamp) / interval);
  const newTokenCount = Math.min(tokenCount + increase, bucketCapacity);
  const newTimestamp =
    newTokenCount < bucketCapacity ? timestamp + interval * increase :
        now;
  return { tokenCount: newTokenCount, timestamp: newTimestamp };
}

function take(oldState, options, now) {
  const { tokenCount, timestamp } = oldState
    ? update(oldState, options, now)
    : { tokenCount: options.bucketCapacity, timestamp: now };
  if (tokenCount > 0 && now >= timestamp) {
    // if there is a token available and the timestamp is in the past
    // take the token and leave the timestamp un-changed
    return { tokenCount: tokenCount - 1, timestamp };
  }
  // update the timestamp to a time when a token will be available,
      leaving
  // the token count at 0
  return { tokenCount, timestamp: timestamp + options.interval };
}

export default async function takeToken(key, options) {
  const now = Date.now();
  const oldState = await db.getRateLimitState(key);
  const newState = take(oldState, options, now);
  // N.B. replaceRateLimitState should throw if current state
  // doesn't match oldState to avoid concurrent token usage
  await db.replaceRateLimitState(key, newState, oldState);
  if (newState.timestamp - now > 0) {
    await new Promise(r => setTimeout(r, newState.timestamp - now));
  }
}
```

Code Snippet B.3: Rate limiting

This code example was taken from Gitconnected[3], since the code found in the manual analysis was too cluttered and obfuscated for the purpose of showing the limitation of user requests.

Discussed by (a.o.): [BMV20] and [MD11].

Description: Rate limiting is a common strategy used to limit the traffic on a website. As the name suggests, the amount of requests in a certain timeframe is limited.

[3] https://levelup.gitconnected.com/rate-limiting-a0783293026a

It puts a limit on how often someone can repeat an action within a certain time-frame – for instance, trying to log in to an account. Rate limiting can help stop certain kinds of malicious bot activity. It can also reduce strain on web servers. However, rate limiting is not a complete solution for managing bot activity.

Countermeasure: Adding delays to make sure the increased load of the server is unnoticeable whenever it is closely monitored. Besides that make sure the delays have random waits in between the requests so that it is even less suspicious.

B.0.4 Honeypot trap (e.g. display:none accessed)

```
1 CSS
2 .dispnon{display: none}
3
4 HTML
5 <input class="dispnon" name="field_name" type="text">
```

Code Snippet B.4: Honeypot trap

Discussed by (a.o.): [SA18].

Description: A honeypot trap is designed to lure bots into revealing the fact that they are not human users. Human users will not be able to see and access these elements physically. However, these elements can be seen by a bot and if the bot is not configured to circumvent honeypot traps, they will reveal themselves.

Countermeasure: The only way to counter honeypot traps is to configure the bots in such a way that they do not access invisible elements or elements that are out of bounds.

B.0.5 Site traversal

Discussed by (a.o.): [TCK09] and [SVA11].

Description: server-side logs contain data on the user navigation patterns of a site. This clickstream data can be analysed and irregularities between how a human user would likely traverse the website and the user in question can lead to a user being flagged as a potential bot. However, this is not a simple task as the amount of logs is immense. Besides that, the logs are all very detailed, which makes it hard to identify the logs which contain the data of the site traversal.

Countermeasure: A human-like path traversal configuration needs to be used in order for the bot not to stand out during the analysis of the site traversal. This can be done by for example taking human site traversal cases as a training set and using machine learning to make sure the bot displays human-like site traversal instead of irregularly jumping all of the place.

Appendix C

Local script analysis

Figure C.1: Time-stamp alert

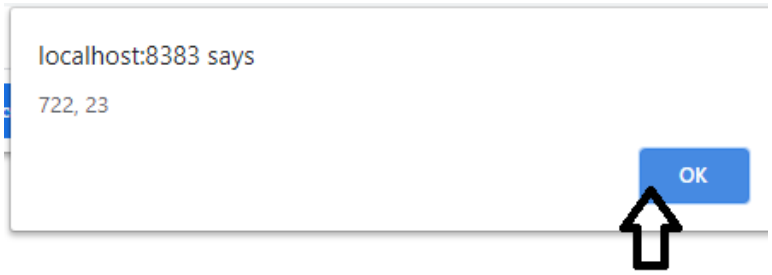


Figure C.2: First mouse position alert

Figure C.3: Second mouse position alert

In the images above, we have drawn arrows as to where the pointer was located at the moment of the alert, since it was invisible on the screenshots taken. We can see the coordinates change according to the position of the pointer. Besides the time tamps and the position of the cursor, the click stream data is also being monitored. Meaning, any object that is clicked on is being recorded. In the images below we can clearly see how the alert shows us both clicking on a DIV element and a HTML element respectively.

Figure C.4: Clicking ⟨div⟩ element alert

Figure C.5: Clicking ⟨html⟩ element alert