This is the *accepted* version of the paper. The final version of the paper can be found at https://doi.org/10.1145/3538969.3538994

To cite this work: Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilios Katos, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2022. Web Bot Detection Evasion Using Deep Reinforcement Learning. In Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES '22). Association for Computing Machinery, New York, NY, USA, Article 15, 1–10. https://doi.org/10.1145/3538969.3538994

Web Bot Detection Evasion Using Deep Reinforcement Learning

Christos Iliou
Information Technologies Institute, CERTH
Thessaloniki, Greece
BU-CERT, Bournemouth University
Bournemouth, United Kingdom
iliouchristos@iti.gr

Theodoros Kostoulas
Department of Information & Communication Systems Engineering, University of the Aegean
Samos, Greece
theodoros.kostoulas@aegean.gr

Theodora Tsikrika
Information Technologies Institute, CERTH
Thessaloniki, Greece
theodora.tsikrika@iti.gr

Vasilis Katos
BU-CERT, Bournemouth University
Bournemouth, United Kingdom
vkatos@bournemouth.ac.uk

Stefanos Vrochidis
Information Technologies Institute, CERTH
Thessaloniki, Greece
stefanos@iti.gr

Ioannis Kompatsiaris
Information Technologies Institute, CERTH
Thessaloniki, Greece
ikom@iti.gr

ABSTRACT

Web bots are vital for the web as they can be used to automate several actions, some of which would have otherwise been impossible or very time consuming. These actions can be benign, such as website testing and web indexing, or malicious, such as unauthorised content scraping, scalping, vulnerability scanning, and more. To detect malicious web bots, recent approaches examine the visitors' fingerprint and behaviour. For the latter, several values (i.e., features) are usually extracted from visitors' web logs and used as input to train machine learning models. In this research we show that web bots can use recent advances in machine learning, and, more specifically, Reinforcement Learning (RL), to effectively evade behaviour-based detection techniques. To evaluate these evasive bots, we examine (i) how well they can evade a pre-trained bot detection framework, (ii) how well they can still evade detection after the detection framework is re-trained on new behaviours generated from the evasive web bots, and (iii) how bots perform if re-trained again on the re-trained detection framework. We show that web bots can repeatedly evade detection and adapt to the re-trained detection framework to showcase the importance of considering such types of bots when designing web bot detection frameworks.

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**; **Supervised learning by classification**; • **Information systems** → *Web log analysis.*

KEYWORDS

advanced web bots, web bot detection, web logs, reinforcement learning, evasive web bots

ARES 2022, August 23–26, 2022, Vienna, Austria

ACM ISBN 978-1-4503-9670-7/22/08…$15.00
https://doi.org/10.1145/3538969.3538994

ACM Reference Format:
Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2022. Web Bot Detection Evasion Using Deep Reinforcement Learning. In *The 17th International Conference on Availability, Reliability and Security (ARES 2022), August 23–26, 2022, Vienna, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3538969.3538994

1 INTRODUCTION

Web bots allow the automation of several vital tasks such as web indexing and website testing, which would have otherwise been very time consuming or impossible. Some tasks require web bots to visit web servers repeatedly, resulting in web bots generating a huge amount of web traffic; based on Imperva's bad bot report [19], web bots accounted for 40.8% of the total traffic that they monitored. Web bots are also used by malicious actors to facilitate attacks [15]. Examples of malicious actions include unauthorised content and price scraping, buying all the available stock of specific limited products to resell at higher price (i.e., scalper bots), vulnerability scanning, brute forcing passwords and credit card numbers, and generating accounts to spam messages or amplify propaganda.

Thus, web servers employ web bot detection techniques, with the most common ones in the last years being based on CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) challenges [35]. However, several approaches have been proposed to bypass various types of CAPTCHAs proposed over the years. Such techniques utilise machine learning [10] along with services such as image reverse image and image tagging [30] to solve CAPTCHA's visual challenges, or speech-to-text engines to solve the audio challenge accompanying CAPTCHAs [7].

To this end, well-known companies that offer web bot detection products [2, 12, 20] as well the latest version of Google's CAPTCHA challenge[1] (reCAPTCHA version 3) examine visitors' fingerprint and behaviour [3] to decide whether a visitor is a bot. Concerning the fingerprint, detection mechanisms collect hardware and software information about the visitors' devices to identify whether they are automated scripts or real browsers. Examples of such techniques are plugin enumeration, WebGL fingerprinting, examination

[1] https://www.google.com/recaptcha

of unique to browser automation software strings in JavaScript variables [6, 22], extraction of browser low level properties such as the instruction-set architecture and the memory allocator used [27].

Even though detecting web bots based on their fingerprint has proven to be very effective against detecting different types of web bots, recent advances in browsing automation software can be used to bypass such techniques. For example, browsing automation software can be specially configured [6, 34] and there are also plugins that can be added to increase the bots' evasiveness, such as the Puppeteer stealth plugin[2] that has been designed to make the Puppeteer browsing automation software[3] harder to detect. Additionally, common browsers can be utilised instead of browsing automation software so as to exhibit a fingerprint indistinguishable from browsers [3]. Thus, to effectively detect web bots, current research examines, besides their fingerprint, the browsing behaviour of the visitors using machine learning based techniques [15, 17, 21, 24].

In this work, we investigate whether advanced web bot that utilise Reinforcement Learning (RL) can evade state-of-the-art web bot detection approaches that analyse web logs. Even though RL has shown very promising results in different scenarios with similar characteristics as the web bot detection/evasion problem, to the best of our knowledge this is the first work that addresses this research question. Additionally, this work considers a scenario closer to a real-world setting, where both the evasive web bots and the web bot detection framework update their methods based on their adversaries' actions; the web bots continuously try to update their behaviour to evade detection, whereas the detection framework updates its models using the evasive behaviours generated by web bots.

The main contributions of this work are:

- The proposal of a novel way to create web bots that use RL to evade detection based on the web logs they generate
- The proposal of a novel RL environment for the web bot detection/evasion problem
- A more realistic evaluation process, where the web bots continuously update their behaviour to evade detection while the detection framework updates its detection models based on the new behaviours generated by the web bots

Overall, this research aims to showcase the possibility of web bots using RL to evade detection effectively. Thus, in this research we aim to show the importance of considering this type of malicious web bots when designing detection frameworks.

The rest of this paper is structured as follows: Section 2 outlines the literature on the behaviour-based web bot detection and detect evasion techniques. Section 3 presents the main RL concepts and how they map to the web bot detection/evasion problem. Section 4 presents the details of the RL environment and the novel evasive web bots that use RL. Section 5 describes the evaluation methodology and the experimental setup, while Section 6 discusses the results. Finally, Section 7 concludes our work and outlines the future work.

[2]https://www.npmjs.com/package/puppeteer-extra-plugin-stealth
[3]https://pptr.dev/

2 RELATED WORK

The web bot detection problem aims to either distinguish web bots from human visitors [8, 11, 29], or to categorise bots based on their functionality [13], purpose [38], or complexity [15, 17]. Next, we initially present the techniques proposed in literature for detecting web bots based on their behaviour (Section 2.1), and then we outline techniques proposed for evading detection (Section 2.2).

2.1 Behaviour-Based Web Bot Detection

Behaviour-based web bot detection techniques commonly examine the web logs of the visitors. Web logs are grouped into sessions and, based on these sessions, several measurable values (i.e., features) are extracted and used to train machine learning models. The trained models use the web logs generated by new visitors to classify them as bots or humans.

Web logs are typically split per user either by using the PHP session ID [16] or, alternatively, based on a combination of the visitor's IP and user agent. Based on those, a unique identifier per visitor is created [14, 26, 29]. Then, for each user, one or more sessions are generated based on a timeout value (i.e., when a specific amount of time have passed and no new requests with the same session identifier have been performed) [14, 15, 26, 29]. Finally, only sessions with a number of requests greater than a threshold can be used, to make sure that the sessions have enough data [26].

After splitting the logs into sessions, the feature extraction process takes place, where several measurable values (i.e., features) are extracted from the web logs. Such features include the total number of requests and the HTTP type of requests (e.g., GET, POST, etc.) [29, 38], the HTTP response code type (e.g., 3xx, 4xx) [8, 38], the percentage of requests to specific file types (e.g., image, CSS, JavaScript) [26], and whether the requests are consecutive (i.e., requests whose URL contains the previously requested URL as a subpart) [38]. Additionally, time-related features, such as session time, browsing speed, inter-request times, etc. [38], have also been considered. Finally, features that capture the semantics of the content of the requested resources, such as the total topics of the pages that the web bot visited and the page similarity and variance, are used [21].

The extracted features are used as input to machine learning algorithms to generate detection models that classify new visitors as bots or humans. For that, usually classification [8, 24] or clustering [25, 32] machine learning algorithms are used. Additionally, the detection process can be either offline after the end of a session [32], or online by performing an estimation during the session [11, 24].

For classification, algorithms that have been used include Support Vector Machines (SVM) [1, 15, 17, 24, 31], Random Forest (RF) [15, 17, 29], MultiLayer Perceptron (MLP) [8, 15, 17, 24, 31], Adaboost [15, 17, 29], k-Nearest Neighbour (kNN) [31], C4.5 [29, 31], Naive Bayes [29], and other Bayesian approaches [33]. An ensemble classifier over different classifiers has also been used [15, 29].

The clustering algorithms that have been used include Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [37], K-means [24, 25], Granded Possibilistic c Means (GPCM) fuzzy clustering [24, 25], unsupervised neural network learning, specifically the Self-Organizing Map (SOM) and the Modified Adaptive Resonance Theory 2 (Modified ART2) [32], SOM in combination with

Fuzzy Rough Set (FRS) theory [14], Markov Clustering (MCL) in combination with FRS theory [38], and Hierarchical Particle Swarm Optimization (HPSO), a clustering approach for outlier detection based on Particle Swarm Optimization (PSO) [4].

Even though most approaches examine the visitors' web logs, there have also been a few approaches proposed in recent years that examine visitors' mouse movements instead. Such approaches extract high level actions from mouse movements, such as click, point-and-click, and drag-and-drop, and then extract features from each action, such as duration, distance, displacement, etc. which are used to train machine learning algorithms [11]. Other recent approaches process the mouse trajectories as images and use them as input to Convolutional Neural Networks (CNNs) [15, 36].

2.2 Behaviour-Based Web Bot Detection Evasion

Web bots proposed in literature commonly use heuristics and statistical distributions to exhibit behaviours that can evade detection based on either the web logs that they generate [9, 18], or their mouse movements [1, 15]. Additionally, more advanced machine learning techniques have been proposed for evading detection based on the mouse movements that bots generate, such as the use of RL to generate mouse movements that can bypass Google reCAPTCHA v3 [3], and the use of Generative Adversarial Networks (GANs) to generate humanlike mouse movements for browsing the web [16]. Finally, GANs have also been used to generate synthetic swipe and accelerometer data for the case of mobile web bots [1].

In this work, we leverage the exceptional performance of RL in different domains (including defeating professional players in games, such as the board game Go [28]), to propose a novel way of creating web bots that can evade detection based on the logs that they generate. Additionally, we use a novel and realistic dynamic environment, where both parties (i.e., the evasive web bots, and the detection framework) update their methods to achieve their goals; the web bots continuously try to find behaviours to evade detection, whereas, the detection framework re-trains its models to detect the new evasive web bots based on the logs they generate.

3 RL MAIN CONCEPTS FOR THE WEB BOT DETECTION/EVASION PROBLEM

In RL, the agent performs actions in an environment in order to maximise the notion of cumulative reward. There are five main "concepts" in RL that should be mapped to the web bot detection/evasion problem: *agent*, *environment*, *action*, *state*, and *reward*. Following an intuitive approach, we map those concepts to the web bot detection/evasion problem, as presented in Figure 1.

Specifically, the web bots repeatedly visit the web server and download content following different behaviours, while also examining whether they appear to have been detected. Based on whether they have been detected or not, web bots update their browsing behaviour. In Table 1 we summarise these main RL concepts and how we instantiated them in the web bot detection/evasion problem.

Furthermore, we assume that web bots can change their fingerprint when being detected, thus starting in a clean-slate every time (as far as the web server is concerned). This is not an unreasonable assumption, since advanced web bots may have several unique fingerprints in their possession allowing them to change their fingerprint and be considered as new visitors by the web server if needed [19].

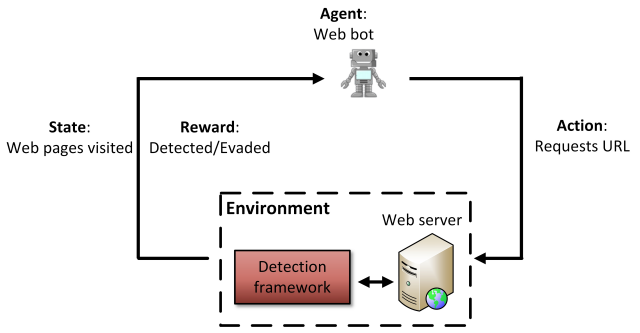

Figure 1: Mapping of the RL concepts to the web bot detection /evasion problem

4 EVASIVE WEB BOTS USING RL

Web bots can take advantage of RL (which has shown very promising results in similar domains) to find behaviours that can evade detection. For that, we initially have to define (i) the appropriate environment (i.e., the web server with the bot detection framework), and then model our evasive web bots by defining (ii) the possible *actions* that web bots can perform, (iii) the *states* the web bots can be in, (iv) the *rewards* that the web bots will receive, and (v) the *model* (i.e., "brain") of the *agent* (i.e., the web bot), responsible for deciding which action to perform at each state.

4.1 Environment

The environment consists of the web server along with the web bot detection framework. The web server hosts several web pages, simulating a real one. The detection framework that focuses on detecting web bots based on their browsing behaviour (i.e., the pages they visit) follows state-of-the-art approaches that examine visitors' web logs [14, 15, 38]. The architecture of the environment is presented in Figure 2.

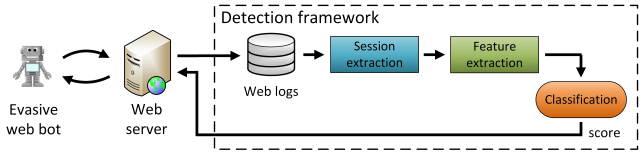

Figure 2: Web server and detection framework

The detection framework periodically accesses the web logs of the web server, and extracts the user sessions from those logs and the values of the features from each session. In the training period, the feature vectors are used as input to train classifiers. Then, the trained classifiers are used to label new visitors as web bots or humans. Next, we present the details of this process.

4.1.1 Session Extraction. The detection framework initially splits the web logs into different visitors and extracts the different sessions

Table 1: RL concepts and instantiation into the web bot detection/evasion problem

RL concept	Description	Web bot detection/evasion context
Agent	The "thing" that senses the environment and performs some actions	Web bot
Environment	The real or simulated "world" that the agent can interact with	Web server and detection framework
Action	What an agent can do in its environment	Visiting of a website
State	Different configurations of the environment that the agent can sense	The web pages that the agent has already visited
Reward	A numerical value received by the agent from the environment as a direct response to the agent's actions	Value that depends on the web page visited and whether the web bot is detected or not

for each visitor. To uniquely identify a user, the PHP session ID is used, while a session ends when more than 30 minutes have passed and no new requests with its ID have been performed [14, 15].

4.1.2 Feature Extraction. For each session, several measurable values (i.e., features) are calculated. These features are related to the method/response code of the HTTP request, the type of file(s) requested, and the browsing behaviour. The features utilised are presented in Table 2 and include the ones from state-of-the-art web bot detection methods proposed in literature that are applicable to our environment [14, 15, 38].

4.1.3 Classification. The extracted feature vectors are used as input to the classifiers. In the training phase, the feature vectors are used to train the classifiers, while in the testing phase, the trained classifiers take as input the feature vectors and decide whether the sessions of those vectors came from bots. The classification module follows state-of-the-art approaches in the web bot detection domain [15], and uses an ensemble (i.e., it performs a class probability averaging of all the available classifiers) of four well known classifiers: SVM, RF, AdaBoost, and MLP Classifier; details of the classifiers that are employed in this work are presented in Section 5.3.1.

4.2 Actions

The actions that the bots will be able to do should depend on what features the state-of-the-art web bot detection considers. As discussed before, these features examine the method/response code of the HTTP request, the type of file(s) requested, and the browsing behaviour (including the pages visited and the time between requests) [17].

Thus, the different actions that are considered in this work have to do with (i) what web pages the bot can visit, and (ii) how much time it spends on them. Based on that and since web bots try to generate a behaviour similar to a human one, the proposed advanced web bots support the following general types of actions:

- *Simple download*: The web bot downloads only the main web page content (e.g., the HTML code) and not the additional files included (e.g., images, CSS, JavaScript, etc.).
- *Full download*: The web bot downloads the web page content and any additional files included (e.g., images, CSS, JavaScript, etc.).

Additionally, before going to the next web page, web bots can wait some time to simulate a "reading" functionality. We consider this waiting time to be between two values, the $time\_min$ and $time\_max$. Also, we consider those to be integers indicating the seconds that web bots will wait, since also the respective features used by the web bot detection framework round the time in seconds, as shown in Table 2.

Thus, we end up with a total number of different actions at each state to be:

$$total\_number\_of\_actions = N \cdot 2 \cdot (dt + 1) \quad (1)$$

where N is the number of web pages of the web server, the '2' indicates the two modes of downloading (i.e., *simple download*, and *full download*), and $dt = time\_max - time\_min$ corresponds to the possible times that the web bot can "wait" on that web page (this is why we add one to the difference).

From Equation 1 we see that web bots can visit any web page when being on a specific web page (and not only the ones included in this web page). If we had allowed web bots to visit web pages only included in the current page, this would not have reflected a scenario closer to a real-world setting, where visitors can access any web page of the server at any time. However, this assumes that web bots know all the URLs of the web pages of the web server. Assuming that web bots will repeatedly visit different web pages of the web server and based on the fact that the URLs of some web servers can be predicted or extracted using different methods (such as using search engines to gather relevant URLs), this is not an unreasonable assumption.

4.3 States

States are considered as the different configurations of the environment that the web bot can sense, and, in our case, are calculated based on the web pages that the bots visit. Each action can result in changing the current state of the agent (i.e., web bot).

There are two ways to calculate the state in our case: (i) considering only the first time a web bot visits a specific web page, or (ii) considering every time a web bot visits a specific web page. The first approach would limit the intelligence of the agent by omitting information that has to do with going back and forth to the same web pages (something that humans usually do). The second case could result in an infinite number of possible states that the web bot can be in, since a web bot can visit a web page infinite times.

Thus, in our case we follow the second approach (i.e., considering every time a web bot visits a specific web page) by adding an upper limit in how many times we consider visits to the same web page. After that, we ignore any additional visits to this page, which could have resulted in an infinite number of possible states.

Additionally, a web bot can simply visit a web page (which is common for simple scripts), or download the additional sources included in this page (such as JavaScript files, CSS files, images,

Table 2: Features extracted from each session

Id	Feature	Short description and literature
1	Total requests	Total number of HTTP requests issued during the session [4, 17, 29, 31, 32, 38]
2	Total session Bytes	Sum of all requested pages' size (in Bytes) in a session [4, 8, 17, 24, 32, 38]
3	HTTP GET requests	Total number of HTTP GET requests issued during the session [5, 8, 17, 29, 38]
4	HTTP POST requests	Total number of HTTP POST requests issued during the session [8, 17, 29, 38]
5	HTTP HEAD requests	Total number of HTTP HEAD requests issued during the session [8, 17, 24, 29, 31, 32, 38]
6	% HTTP 3xx requests	Percentage of HTTP requests that led to an HTTP 3xx code response [5, 8, 17, 38]
7	% HTTP 4xx requests	Percentage of HTTP requests that led to an HTTP 4xx code response [5, 8, 17, 24, 31, 32, 38]
8	% image requests	Percentage of HTTP requests that requested an image. This feature searches for all known image formats' ending [17, 26, 29]
9	% css file request	Percentage of HTTP requests that requested a css file [17, 26]
10	% js requests	Percentage of HTTP requests that requested a JavaScript file [17, 26]
11	HTML-to-image ratio	The number of the requested HTML files divided by the number of requested image files in a session [17, 31, 38]
12	Depth SD	Standard deviation of requested pages' depth (i.e. number of '/' in URL path) [17, 31, 32, 38]
13	Max requests per page	Maximum number of requests to the same page in a session [17]
14	Average requests per page	Average number of requests per page in a session [17]
15	Max number of consecutive sequential HTTP requests	Maximum number of HTTP requested URLs that contain the previously requested URL as a subpart page [17, 38]
16	% of consecutive sequential HTTP requests	Percentage of HTTP requested URLs that contain the previously requested URL as a subpart [17, 31, 32]
17	Session time	Total time (in seconds) between the first and the last HTTP request of the session [4, 5, 17, 24, 29, 38]
18	Browsing speed	Ratio of the total number of requested pages over time (in seconds) [5, 17]
19	SD of inter-request times	Standard deviation of time between successive requests [5, 17]

etc.). In the case of human visitors (i.e., where a browser is used), the latter (i.e., downloading additional sources included in the web page) is usually done only in the first time each file is encountered and the downloaded files are cached on the browser.

Thus, in our setup we consider two separate states for each web page based on the way the web page can be visited: (i) only the web page is downloaded, and (ii) the web page is downloaded along with the additional files included in that page.

We define the state vector for a web server with N web pages as:

$$state\_vector = [\#page_0, ..., \#page_{N-1}, \#page_N, ..., \#page_{2N-1}] \quad (2)$$

where $\#page_i$ is an integer number indicating how many times the agent has visited a specific web page $page_i$, getting values between 0 and M, with M being the upper limit for the number of times that we consider that the web bot can visit the specific web page. Additionally, web pages whose index differs by N (i.e., $page_i$ and $page_{N+i}$, $0 \leq i \leq N-1$) correspond to the same web page, with the one with the low index indicating that only the web page is downloaded while the one with the high index indicating that, besides the web page, the additional files included in the web page (e.g., JavaScript, CSS, etc.) are also downloaded.

4.4 Rewards

Web bots can be trained to maximise a notion of cumulative reward, received by the environment as a direct response to their actions. The selection of rewards is very important, since the rewards will guide the web bot in achieving the wanted behaviour. For example, simply giving positive rewards to the web bot by not being detected might result in the web bot staying in the same web page, as long as it remains undetected.

In our case, the target goal of the web bots is twofold: (i) to generate a behaviour that evades detection, and (ii) to explore the web server and visit new web pages. Thus, we consider the following types of rewards:

- *New web page reward:* Web bot visits a new web page and does not get detected
- *New state reward:* Web bot changes state (but does not visit a web page that has not visited before) and does not get detected. As discussed before, a web bot can visit the same web page up to M times resulting in another state each time
- *Detection reward:* Web bot tries to download any web page and gets detected
- *Detection evasion reward:* The web bot manages to visit M web pages and not get detected.

Based on the above, the *detection reward* should be the lowest, motivating the bot to select the actions that will allow it to remain undetected. This reward can also be negative, resulting in a "penalising" characteristic. The *new web page reward* should be a positive one, to motivate the web bot to navigate the web server and visit different web pages. The *new state reward* can also be a positive one since the web bot might have to re-visit a specific web page to evade detection since this is also something that humans do. But it should be lower than the *new web page reward*, since otherwise web bots might prefer to stay at the same page as long as they do

not get detected. Finally, the *detection evasion reward* should be the highest one, motivating web bots to follow specific behaviours that will allow it to evade detection.

4.5 Agent

After having defined the environment, actions, states, and rewards, we have to choose the algorithm that the agents (i.e., web bots) should follow to learn what action they should perform at each state.

We decided to base our *model* to the well-known Q-learning algorithm. Q-learning generates a matrix containing all possible states and the respective actions and a value for each state-action pair representing how useful a specific action is on a specific state based on the future reward that the agent will receive. This is called the action-value function, $Q(s, a)$. Using Q-learning, the web bot can find a *policy* (i.e., the way it makes decisions for what actions to perform at each state) to follow that maximises its future reward.

The $Q(s, a)$ is initialised randomly at the beginning. Then, for each step t the agent takes an action a_t on a state s_t resulting in changing its state to s_{t+1} and receiving a reward r_t. Based on this transition, the $Q(s, a)$ is updated using the following equation.

$$Q^{\text{new}}(s_t, a_t) = Q(s_t, a_t) + a \cdot \left(r_t + \gamma \cdot \max Q(s_{t+1}, a) - Q(s_t, a_t) \right) \quad (3)$$

where the a (i.e., learning rate) determines to what extent the newly acquired information overrides old information, and the γ (i.e., discount factor) determines the importance of future rewards. In this, the Temporal Difference (TD) learning approach is followed, where the agent tries an action (a_t) in a particular state (s_t), and evaluates its consequences in terms of the immediate reward and its estimate of the value of future rewards it will receive by moving to the next state, s_{t+1}.

As discussed above, to calculate $Q(s, a)$, the agent (i.e., web bot) interacts with the environment and updates $Q(s, a)$ based on the states it has been in and the actions it performs. However, by following only the most promising actions in a state, the agent chooses only the actions that give the highest immediate reward, which might be far less than the future reward that it could have received by performing different actions (which are currently unknown).

Thus, the agent should find a balance between "exploitation" (i.e., performing the most promising action), and "exploration" (i.e taking a random action, hoping for a better future reward). This is called the "explore-exploit" dilemma, where the agent has to decide when to explore and when to exploit (i.e., take the best action based on the current knowledge of the environment). In principle, the agent could have always performed random actions to find the best behaviour to follow, but this would take a lot of time due to the agent following paths that are not very promising.

To calculate the $Q(s, a)$, our agent uses the Epsilon-Greedy approach, where it performs a random action or the most "promising" action on a current state based on a probability ϵ. Based on this probability, the agent either (i) follows the most promising states (in regards of the future rewards that it will receive), or (ii) performs random actions that may result in states with better future rewards.

To account for the fact that by following the Q-learning algorithm as presented above, the agent might take a lot of time to train (since there is a large number of actions the agent can perform, as shown in Equation 1), in this work we used the Deep Q-Network (DQN) algorithm proposed by DeepMind [23]. DQN uses a Deep Neural Network (DNN) to calculate the equivalent of the $Q(s, a)$ used in Q-learning that allows us to train the web bots faster. Thus, instead of calculating the $Q(s, a)$, we calculate an approximation using DNNs.

Specifically, in DQN, we approximate the $Q(s, a)$ by considering a network that takes as input the current state vector and have multiple output nodes, each one representing the value for each different action (i.e., the output layer of the neural network has the same size as the number of possible actions). Additionally, we use the experience replay technique, where we store all of the agent's experiences and then randomly sample from these experiences (i.e., get a random minibatch of the transitions) and use the samples to perform a gradient descent to train the DQN.

As it is evident, the DQN architecture depends on the number of states (which depends on the web pages considered, as shown in Equation 2), and the total number of actions (which depends on the web pages and the waiting time, as shown in Equation 1). Since these depend on the experimental setup, the architecture of the DQN is presented in Section 5.3.

5 EVALUATION

To assess the effectiveness of the web bots that use RL to evade detection, we consider the case of scraping web bots that continuously visit a web server to harvest its content. The web server uses a web bot detection framework that examines the visitors web logs to identify and block bot visitors.

5.1 Evaluation Methodology

To simulate a more realistic scenario (where both the evasive web bots and the detection framework continuously update their techniques to achieve their goals), the evaluation takes place in two phases. In the first phase, we assume that the detection framework knows nothing about the web bots that use RL, while in the second the detection framework is trained on the behaviours of those bots.

In the first phase of the experiments, we used a dataset on the web bot detection domain [15], which includes both human and bot sessions, with the latter being generated from bots that use heuristics to evade detection. We opted to do that, to see how the web bots using RL can be trained to evade a web bot detection framework that was trained on different types of web bots.

In the second phase, we use data collected during the first phase from the web bots that used RL and successfully evaded detection to re-train the detection framework and evaluate how well these web bots can still evade detection. Additionally, we examine the case of web bots re-training their models to the re-trained detection framework. We argue that this setup simulates a scenario closer to a real-world setting, where both the defender (i.e., the web bot detection framework) and the attacker (i.e., the evasive web bots) continuously update their models.

Moreover, since in a realistic scenario we want to detect web bots as soon as possible, the detection framework examines the behaviour of the web bots per-request, and uses different classification models for each number of requests considered. Thus, to

train the detection framework, we performed an iterative process where we initially considered only the first request in each session and gradually increased the number of requests considered. When a session reaches the maximum number of requests available, we stop increasing the number of requests considered for that session.

In both phases of the experiments, we performed the same train-test split (with about 70% of the data considered as training and the rest as testing) on the visitors when evaluating the detection framework (i.e., we split the data based on the visitors to group multiple sessions from the same visitor in the same set). Additionally, at each request considered we performed a grid search over several possible values of hyperparameters on the (current) training set and select the best performing ones using a 2-fold cross validation. Thus, each classifier uses a unique set of parameters for each number of requests considered. This increases the training time considerably but usually increases the performance.

Since in RL the agents (in our case the web bots) can be trained indefinitely, we evaluated them at different training times (i.e., at different numbers of requests performed cumulatively by all web bots). We opted to do that because when a bot is detected, it is blocked and will not be able to revisit the web server (at least temporarily). Thus, there is a trade-off between how much training the web bots should do (in regards to the bots required and the time) and the evasiveness of the behaviour generated.

Finally, besides the web bots that use RL to evade detection, we also tested other types of web bots that use heuristics. We did that for comparison purposes and because of different approaches proposed in literature on evasive web bots that also use heuristics to evade detection [9, 18]. The details of the configurations of the web bots considered are presented in Section 5.3.3.

5.2 Dataset

To evaluate our web bots, we used the web server and dataset from the work of [15]. The web server used in [15] hosted a total of 110 web pages from 11 categories/topics crawled from Wikipedia[4].

Using this web server, in [15] a total of 56 human sessions and 56 bot sessions were created. For the human sessions, 28 human subjects visited the web server (with no strict instructions on how to read the content or what content to read to simulate a more realistic scenario), and created 2 sessions of 15-20 minutes per session each. To generate the web bot sessions, the authors from [15] used the *advanced web bots* that they proposed. These *advanced web bots* browsed the web server following hyperlinks in a humanlike manner, i.e., they were more likely to follow hyperlinks from within the same topic (which is also usually done by humans), and also emulated a "reading" functionality by spending more time, with a probability, on specific web pages. The time depends on the size of each web page.

The aforementioned dataset containing these 112 sessions was used in this work. Out of those, 20 humans and 20 bots (80 session in total) were used as the training set (~70%), and the rest used for testing.

[4]https://www.wikipedia.org/

5.3 Bot and Environment Configurations

5.3.1 Web Bot Detection. The web bot detection framework follows state-of-the-art approaches [15] and uses a classifier which ensembles four well established machine learning classifiers: SVM, RF, AdaBoost, and MLP. It performs a class probability averaging of all the above classifiers to increase the effectiveness of the detection framework.

As discussed before, each classifier is trained on each number of requests considered, with the total number of ensemble classifiers being equal to the total number of requests considered. For each classifier, we re-calculate the set of hyperparameters that achieve the highest accuracy using grid search with a 2-fold cross validation. This was done to increase the general effectiveness.

5.3.2 Bots Using Deep RL. As discussed in Section 4, to model the evasive web bots that use RL, we need to define the possible *actions* that web bots can perform, the *states* the web bots can be in, the respective *rewards*, and the *model* that the *agent* (i.e., web bot) will use to decide the actions to perform. The respective configurations are presented below:

Actions: For the actions, we heuristically selected the $time\_min = 2\ sec$ and the $time\_max = 10\ sec$ so that web bots can perform relatively rapid actions (to scrape the server within a short period of time), and because we do not want web bots to perform all actions very fast and end up getting detected.

States: As mentioned in Section 4, a maximum value of the times that we consider that a web page has been visited is calculated, M. Thus, we set $M = 60$, since the maximum number of web pages downloaded by the humans in our dataset [15].

Rewards: The rewards should motivate the web bot to (i) remain undetected, and (ii) visit new states. The rewards used were selected heuristically and are presented in Table 3.

Table 3: Rewards

Action	Detected	Reward
New web page reward	No	+1.0
New state reward	No	+0.1
Detection reward	Yes	0.0
Evasion reward	No	+100.0

Agent: As discussed in Section 4, we use the DQL algorithm, with the input layer size being equal to the the number of states, and the output layer size being equal to the number of possible actions. The rest of the architecture was heuristically selected and is presented below.

Table 4: DQN architecture

Layer type	Output Shape	Activation
Flatten	2*N = 2*110 = 220	–
Dense	1024	ReLU
Dense	2*N*(dt+1) = 2*110*(8+1) = 1980	Linear

For the implementation of the environment, the Gym[5] Python library was used, that provides standard APIs allowing the communication between the RL algorithms and the environments. For the implementation of the DQN, the keras-rl2[6] python library was used, which implements state-of-the art Deep RL algorithms, seamlessly integrates with the Keras[7] deep learning library, and works out of the box with OpenAI Gym.

The hyperparameters for the DQN were selected heuristically and are presented in Table 5. The ϵ was selected to be 0.2, so that the web bots follow the behaviours that have found to currently be the more evasive ones, but at the same time, with a smaller probability, try to find new promising behaviours (i.e., exploring). The γ was set to 0.9 enabling bots to focus on future rewards. For the other hyperparameters, values that are commonly used were selected.

Table 5: Parameters and configurations

Parameter	Value
Warm up steps	100
Learning rate	10^{-3}
Epsilon greedy probability (ϵ)	0.2
Discount factor (γ)	0.9
Replay Memory	size=50k, window=1

Finally, since we want to evaluate how well the web bots perform when placed into different web pages as a starting point in testing (and not choosing the most evasive starting point each time), each bot performs a random step at the beginning[8].

5.3.3 Bots Using Heuristics. In addition to the web bots that use RL to evade detection, we used some additional types of web bots that use heuristics to evade detection, mainly for comparison purposes. These bots are detailed below:

- *Simple download bot:* Web bot that performs only *simple download* actions, i.e. downloads only the web page content and not the additional files included in the web page (e.g., CSS files, JavaScript files, images, etc.)
- *Full download bot:* Web bot that performs only *full download* actions, i.e. downloads the web page content and additional files included in the web page (e.g., CSS files, JavaScript files, images, etc.)
- *Random bot:* Web bot that downloads either only the web page or the additional files included (i.e., it performs a random action from *simple download* or *full download*)
- *Heuristic bot #1:* For the first request it performs a *full download*, and for the rest of the requests it performs a *simple download*. This simulates the behaviour of a browser, where the additional content of web pages is cached.
- *Heuristic bot #2:* Similar to the *heuristic bot #1*, but in this case the web bot waits for a time between 8∼10 seconds (instead of 2∼10). We used that, since humans usually spend more time on web pages.

[5]https://github.com/openai/gym
[6]https://github.com/taylormcnally/keras-rl2
[7]https://keras.io/
[8]This was not supported by the keras-rl2 library, so we had to update the source code of the library

As discussed before, we assume that all bots start from a random web page and can visit any web page.

5.4 Evaluation Metrics

To evaluate the web bot detection framework, we calculated the (balanced) accuracy, and the precision and recall for both classes (i.e., web bots and humans), all of which are commonly used in the web bot detection domain [15]. To evaluate how well the web bots evaded detection, we calculated the evasion percentage (i.e., how many web bots evaded detection from the total web bots tested).

6 RESULTS

As discussed previously, the evaluation takes place in two phases; in the first phase the web bots that use RL are trained and evaluated on an already trained web bot detection framework, and in the second phase the web bots are evaluated on the detection framework that was re-trained using the new bot behaviours collected in the first phase. In the second phase, we evaluate the bots trained in the first phase and examine the case of web bots also re-training on the re-trained detection framework. Next, we present the results of the two evaluation phases.

6.1 First Evaluation Phase

In the first evaluation phase, we initially evaluate the web bot detection framework that is based on [15] (for completeness purposes), and then evaluate the evasive web bots against this framework.

6.1.1 Web Bot Detection Performance. The performance of the detection framework per request considered is shown in Figure 3. Similarly to [15], the framework can effectively detect web bots; in addition, since those web bots use heuristics to evade detection, it is not always easy to preserve a humanlike behaviour for several requests and, sometimes, specific types or requests may reveal the nature of web bots.

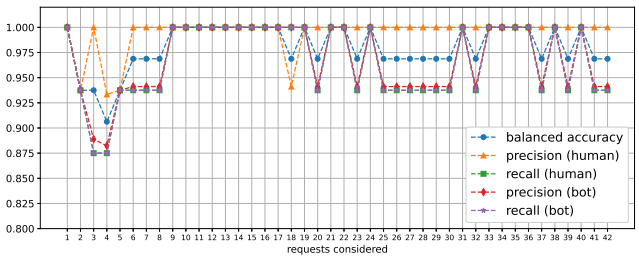

Figure 3: Performance of the web bot detection framework

6.1.2 Evasive Web Bots. Next, we present the results of the proposed evasive web bots by testing 1000 bots of each bot type. For the RL bots, we consider them being initially trained on the pre-trained web bots detection framework, and then using 1000 additional bots for testing. The results are presented in Table 6.

Based on the evasion percentage, we see that the heuristic approaches tested can be easily detected. Only a few web bots, following mainly the *simple download* approach, evaded detection. This can be attributed to the fact that the *advanced web bots* used

Table 6: Performance of evasive web bots

Bot type	Bots / Requests (used for training)	Evasion percentage
Simple download	-	0.7%
Full download	-	0.0%
Random	-	0.0%
Heuristic #1	-	0.1%
Heuristic #2	-	0.0%
RL bot @5k	850 / 5k	0.0%
RL bot @10k	1,721 / 10k	5.5%
RL bot @15k	2,407 / 15k	10.9%
RL bot @20k	3,202 / 20k	6.3%
RL bot @30k	4,893 / 30k	38.5%
RL bot @40k	5,841 / 40k	37.3%
RL bot @50k	5,331 / 50k	36.2%
RL bot @100k	10,528 / 100k	39.3%
RL bot @200k	17,476 / 200k	22.6%
RL bot @300k	**27,175 / 300k**	**44.2%**
RL bot @400k	39,470 / 400k	31.9%
RL bot @500k	41,606 / 500k	38.0%

for training the detection framework performed a *full download* in their first request so as to exhibit a humanlike behaviour.

On the contrary, web bots using RL were able to find behaviours that can evade detection. Generally, the more requests considered in training, the better the evasive web bots perform. However, we see that in some cases the performance of the web bots decreases. This can be attributed to web bots trying ("exploring") new actions to specific states (instead of following the most promising ones) which can result in the decrease of the reward values of those states (resulting in web bots no longer selecting them). However, in the long run, this does not affect the performance of the web bots.

6.2 Second Evaluation Phase

In the second phase, we evaluate the detection framework re-trained on the logs generated by a specific type of RL bot from the first phase, when faced with (i) the same pre-trained RL bot, and (ii) when the RL bots are re-trained on the (now re-trained) detection framework. In this phase, we used the *RL bots @300k* which appeared to be the most evasive ones during the first evaluation phase. Next, we first present the performance of the re-trained web bot detection framework and then we re-evaluate the web bots on the updated framework.

6.2.1 Web Bot Detection Performance. The re-trained web bot detection framework manages to correctly classify the visitors as bots or humans for every request, achieving a 100% accuracy for all requests. This indicates that the web bots that use RL generated a behaviour that, even though evaded detection, can be distinguished from the human behaviour, when known.

6.2.2 Evasive Web Bots. As discussed above, in the second evaluation phase the web bots that use RL are re-evaluated on the re-trained web bot detection server. For that, we consider two cases: (i) the first case where the web bots do not perform any additional training, and (ii) the second case, where web bots re-train to evade the re-trained web bot detection framework. Table 7 presents the results of the second evaluation phase.

Table 7: Performance of evasive web bots on the re-trained detection server

Bot type	Bots / Requests (used for training)	Evasion percentage
RL bot @300k	(trained before)	3.4%
RL bot @5k	199 / 5k	6.4%
RL bot @10k	218 / 10k	1.9%
RL bot @15k	349 / 15k	6.1%
RL bot @20k	947 / 20k	2.1%
RL bot @30k	935 / 30k	3.2%
RL bot @40k	958 / 40k	1.9%
RL bot @50k	1,270 / 50k	2.6%
RL bot @100k	3,488 / 100k	8.0%
RL bot @200k	7,694 / 200k	10.7%
RL bot @300k	**11,875 / 300k**	**18.0%**
RL bot @400k	15,172 / 400k	5.5%
RL bot @500k	18,040 / 500k	8.1%
RL bot @1m	32,476 / 1m	5.2%
RL bot @2m	**65,779 / 2m**	**56.5%**

As expected, evading the re-trained web bot detection server is far more challenging. We see however that some of the pre-trained web bots from the first evaluation phase manage to evade detection. This can be attributed to the fact that the web server used only 56 bot sessions randomly selected for training which probably did not cover all possible behaviours that the web bots can generate.

Additionally, we see that it is much more difficult for web bots to find evasive behaviours against the re-trained detection framework and need additional training requests. Still, we see that the web bots training for the same time as in the first evaluation phase (i.e., 300k requests) manage to evade detection with an adequate percentage. Furthermore, we see that training with additional requests considerably increases the evasiveness of the web bots, and even achieving a higher evasive percentage compared with the first evaluation phase.

When selecting how much we should train our web bots, we have to consider that increasing the training time might not be the optimal approach for the web bot detection domain. For example, for the case of 2 million requests, ~65.8k bots are required, with most of them needing a unique fingerprint (to be able to re-visit with a clean slate after being detected). Considering that the web bot detection framework might choose to update their detection models relatively often, selecting less evasive bots that perform adequately and avoid spending too much time on evading one specific detection framework might be a better choice.

7 CONCLUSIONS AND FUTURE WORK

This work proposed a novel way for bots to evade detection based on the web logs they generate through the use of RL. We evaluated the proposed web bots against (i) a pre-trained detection framework

that was trained on different types of web bots, and (ii) a web bot detection framework that was trained on web logs generated from web bots that use RL. For the latter, we considered two cases, the case of how well the web bots trained on the pre-trained web bot detection framework evade detection, and the case of the evasive web bots re-training on the re-trained detection framework. We followed this approach to simulate a scenario closer to a real-world setting, where both the web bots and the web server update their models to achieve their goals.

The results show that web bots that use RL can adequately evade detection even when the web server knows or has seen their behaviour. This indicates that there is a need for additional detection mechanisms to detect such web bots that use recent advances in machine learning to evade detection. Future work includes the examination of additional methods that use recent advanced machine learning techniques and can be used by web bots to evade detection, as well as techniques tailored to detect them.

ACKNOWLEDGMENTS

This work was supported by the FORESIGHT (H2020 833673), ECHO (H2020 830943), and IDEAL-CITIES (H2020 778229) projects, funded by the European Commission.

REFERENCES

[1] Alejandro Acien, Aythami Morales, Julian Fiérrez, Rubén Vera-Rodríguez, and Oscar Delgado-Mohatar. 2020. BeCAPTCHA: Bot Detection in Smartphone Interaction using Touchscreen Biometrics and Mobile Sensors. *CoRR* abs/2005.13655 (2020). arXiv:2005.13655 https://arxiv.org/abs/2005.13655

[2] Akamai. 2021. Akamai's Bot Manager - Advanced strategies to flexibly manage the long-term business and IT impact of bots. https://www.akamai.com/us/en/multimedia/documents/product-brief/bot-manager-product-brief.pdf

[3] Ismail Akrout, Amal Feriani, and Mohamed Akrout. 2019. Hacking Google reCAPTCHA v3 using Reinforcement Learning. *CoRR* abs/1903.01003 (2019). arXiv:1903.01003 http://arxiv.org/abs/1903.01003

[4] Shafiq Alam, Gillian Dobbie, Yun Sing Koh, and Patricia Riddle. 2014. Web bots detection using Particle Swarm Optimization based clustering. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014*. IEEE, 2955–2962. https://doi.org/10.1109/CEC.2014.6900644

[5] Yasmin AlNoamany, Michele C. Weigle, and Michael L. Nelson. 2013. Access patterns for robots and humans in web archives. In *13th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL '13, Indianapolis, IN, USA, July 22 - 26, 2013*, J. Stephen Downie, Robert H. McDonald, Timothy W. Cole, Robert Sanderson, and Frank Shipman (Eds.). ACM, 339–348. https://doi.org/10.1145/2467696.2467722

[6] Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, and Nick Nikiforakis. 2020. Web Runner 2049: Evaluating Third-Party Anti-bot Services. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24-26, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12223)*, Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves (Eds.). Springer, 135–159. https://doi.org/10.1007/978-3-030-52683-2_7

[7] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. 2017. unCaptcha: A Low-Resource Defeat of reCaptcha's Audio Challenge. In *11th USENIX Workshop on Offensive Technologies, WOOT 2017, Vancouver, BC, Canada, August 14-15, 2017*, William Enck and Collin Mulliner (Eds.). USENIX Association. https://www.usenix.org/conference/woot17/workshop-program/presentation/bock

[8] Alberto Cabri, Grazyna Suchacka, Stefano Rovetta, and Francesco Masulli. 2018. Online Web Bot Detection Using a Sequential Classification Approach. In *20th IEEE International Conference on High Performance Computing and Communications; 16th IEEE International Conference on Smart City; 4th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2018, Exeter, United Kingdom, June 28-30, 2018*. IEEE, 1536–1540. https://doi.org/10.1109/HPCC/SmartCity/DSS.2018.00252

[9] Michele Campobasso, Pavlo Burda, and Luca Allodi. 2019. CARONTE: Crawling Adversarial Resources Over Non-Trusted, High-Profile Environments. In *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 433–442. https://doi.org/10.1109/EuroSPW.2019.00055

[10] Jun Chen, Xiangyang Luo, Yanqing Guo, Yi Zhang, and Daofu Gong. 2017. A Survey on Breaking Technique of Text-Based CAPTCHA. *Secur. Commun. Networks* 2017 (2017), 6898617:1–6898617:15. https://doi.org/10.1155/2017/6898617

[11] Zi Chu, Steven Gianvecchio, and Haining Wang. 2018. Bot or Human? A Behavior-Based Online Bot Detection System. In *From Database to Cyber Security - Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday*. 432–449. https://doi.org/10.1007/978-3-030-04834-1_21

[12] Cloudflare. 2021. Cloudflare Bot Management. https://www.cloudflare.com/en-gb/products/bot-management/

[13] Derek Doran and Swapna S. Gokhale. 2012. A classification framework for web robots. *J. Assoc. Inf. Sci. Technol.* 63, 12 (2012), 2549–2554. https://doi.org/10.1002/asi.22741

[14] Javad Hamidzadeh, Mahdieh Zabihimayvan, and Reza Sadeghi. 2018. Detection of Web site visitors based on fuzzy rough sets. *Soft Comput.* 22, 7 (2018), 2175–2188. https://doi.org/10.1007/s00500-016-2476-4

[15] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2021. Detection of Advanced Web Bots by Combining Web Logs with Mouse Behavioural Biometrics. *Digital Threats: Research and Practice* 2, 3, Article 24 (jun 2021), 26 pages. https://doi.org/10.1145/3447815

[16] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2021. Web Bot Detection Evasion Using Generative Adversarial Networks. In *IEEE International Conference on Cyber Security and Resilience, CSR 2021, Rhodes, Greece, July 26-28, 2021*. IEEE, 115–120. https://doi.org/10.1109/CSR51186.2021.9527915

[17] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Yiannis Kompatsiaris. 2019. Towards a framework for detecting advanced Web bots. In *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*. ACM, 18:1–18:10. https://doi.org/10.1145/3339252.3339267

[18] Christos Iliou, Theodora Tsikrika, Stefanos Vrochidis, and Yiannis Kompatsiaris. 2017. Evasive Focused Crawling by Exploiting Human Browsing Behaviour: a Study on Terrorism-Related Content. In *Proceedings of the 1st International Workshop on Cyber Deviance Detection co-located with the Tenth International Conference on Web Search and Data Mining (CyberDD @ WSDM 2017), Cambridge, UK, February, 10, 2017*.

[19] Imperva. 2021. Bad Bot Report 2021: The Pandemic of the Internet. https://www.imperva.com/blog/bad-bot-report-2021-the-pandemic-of-the-internet/

[20] Imperva. 2021. Data User Behavior Analytics. https://www.imperva.com/products/data-user-behavior-analytics/

[21] Athanasios Lagopoulos and Grigorios Tsoumakas. 2020. Content-aware web robot detection. *Appl. Intell.* 50, 11 (2020), 4017–4028. https://doi.org/10.1007/s10489-020-01754-9

[22] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting: A Survey. *ACM Trans. Web* 14, 2 (2020), 8:1–8:33. https://doi.org/10.1145/3386040

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR* abs/1312.5602 (2013). arXiv:1312.5602 http://arxiv.org/abs/1312.5602

[24] Stefano Rovetta, Alberto Cabri, Francesco Masulli, and Grazyna Suchacka. 2019. Bot or Not? A Case Study on Bot Recognition from Web Session Logs. In *Quantifying and Processing Biomedical and Behavioral Signals*, Anna Esposito, Marcos Faúndez-Zanuy, Francesco Carlo Morabito, and Eros Pasero (Eds.). Smart Innovation, Systems and Technologies, Vol. 103. Springer, 197–206. https://doi.org/10.1007/978-3-319-95095-2_19

[25] Stefano Rovetta, Grazyna Suchacka, and Francesco Masulli. 2020. Bot recognition in a Web store: An approach based on unsupervised learning. *J. Netw. Comput. Appl.* 157 (2020), 102577. https://doi.org/10.1016/j.jnca.2020.102577

[26] Nathan Rude and Derek Doran. 2015. Request Type Prediction for Web Robot and Internet of Things Traffic. In *14th IEEE International Conference on Machine Learning and Applications, ICMLA 2015, Miami, FL, USA, December 9-11, 2015*, Tao Li, Lukasz A. Kurgan, Vasile Palade, Randy Goebel, Andreas Holzinger, Karin Verspoor, and M. Arif Wani (Eds.). IEEE, 995–1000. https://doi.org/10.1109/ICMLA.2015.53

[27] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript Template Attacks: Automatically Inferring Host Information for Targeted Exploits. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society.

[28] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.* 529, 7587 (2016), 484–489. https://doi.org/10.1038/nature16961

[29] Dilip Singh Sisodia, Shrish Verma, and Om Prakash Vyas. 2015. Agglomerative approach for identification and elimination of web robots from web server logs to

extract knowledge about actual visitors. *Journal of Data Analysis and Information Processing* 3, 01 (2015), 1.

[30] Suphannee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. 2016. I am Robot: (Deep) Learning to Break Semantic Image CAPTCHAs. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. 388–403.

[31] Dusan Stevanovic, Aijun An, and Natalija Vlajic. 2012. Feature evaluation for web crawler detection with data mining techniques. *Expert Syst. Appl.* 39, 10 (2012), 8707–8717. https://doi.org/10.1016/j.eswa.2012.01.210

[32] Dusan Stevanovic, Natalija Vlajic, and Aijun An. 2013. Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Appl. Soft Comput.* 13, 1 (2013), 698–708. https://doi.org/10.1016/j.asoc.2012.08.028

[33] Grazyna Suchacka and Mariusz Sobkow. 2015. Detection of Internet robots using a Bayesian approach. In *2nd IEEE International Conference on Cybernetics, CYBCONF 2015, Gdynia, Poland, June 24-26, 2015*. IEEE, 365–370. https://doi.org/10.1109/CYBConf.2015.7175961

[34] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. 2020. FP-Crawlers: studying the resilience of browser fingerprinting to block crawlers. In *MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web*.

[35] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. 2003. CAPTCHA: Using Hard AI Problems for Security. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2656)*, Eli Biham (Ed.). Springer, 294–311. https://doi.org/10.1007/3-540-39200-9_18

[36] Ang Wei, Yuxuan Zhao, and Zhongmin Cai. 2019. A Deep Learning Approach to Web Bot Detection Using Mouse Behavioral Biometrics. In *Biometric Recognition - 14th Chinese Conference, CCBR 2019, Zhuzhou, China, October 12-13, 2019, Proceedings*. 388–395. https://doi.org/10.1007/978-3-030-31456-9_43

[37] Mahdieh Zabihi, Majid Vafaei Jahan, and Javad Hamidzadeh. 2014. A density based clustering approach for web robot detection. In *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 23–28.

[38] Mahdieh Zabihimayvan, Reza Sadeghi, H. Nathan Rude, and Derek Doran. 2017. A soft computing approach for benign and malicious web robot detection. *Expert Syst. Appl.* 87 (2017), 129–140. https://doi.org/10.1016/j.eswa.2017.06.004