Towards a framework for detecting advanced Web bots

Christos Iliou
Information Technologies Institute, CERTH
Thessaloniki, Greece
Dept. of Computing and Informatics, Bournemouth University
Bournemouth, United Kingdom
iliouchristos@iti.gr

Theodoros Kostoulas
Dept. of Computing and Informatics, Bournemouth University
Bournemouth, United Kingdom
tkostoulas@bournemouth.ac.uk

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

Vasilis Katos
Dept. of Computing and Informatics, Bournemouth University
Bournemouth, United Kingdom
vkatos@bournemouth.ac.uk

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

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

ABSTRACT

Automated programs (bots) are responsible for a large percentage of website traffic. These bots can either be used for benign purposes, such as Web indexing, Website monitoring (validation of hyperlinks and HTML code), feed fetching Web content and data extraction for commercial use or for malicious ones, including, but not limited to, content scraping, vulnerability scanning, account takeover, distributed denial of service attacks, marketing fraud, carding and spam. To ensure their security, Web servers try to identify bot sessions and apply special rules to them, such as throttling their requests or delivering different content. The methods currently used for the identification of bots are based either purely on rule-based bot detection techniques or a combination of rule-based and machine learning techniques. While current research has developed highly adequate methods for Web bot detection, these methods' adequacy when faced with Web bots that try to remain undetected hasn't been studied. For this reason, we created and evaluated a Web bot detection framework on its ability to detect conspicuous bots separately from its ability to detect advanced Web bots. We assessed the proposed framework performance using real HTTP traffic from a public Web server. Our experimental results show that the proposed framework has significant ability to detect Web bots that do not try to hide their bot identity using HTTP Web logs (balanced accuracy in a false-positive intolerant server > 95%). However, detecting advanced Web bots that present a browser fingerprint and may present a humanlike behaviour as well is considerably more difficult.

CCS CONCEPTS

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

KEYWORDS

Web bot detection, Evasive Web bots, Advanced Web bots, humanlike behaviour

ACM Reference Format:
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) (ARES '19), August 26–29, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3339252.3339267

1 INTRODUCTION

The vast amount of content hosted on the Internet has rendered the use of Web bots necessary. Web bots are programs that automate the browsing process and perform specific commands on behalf of the author. Popular uses of Web bots include Web indexing, Website monitoring (validation of hyperlinks and HTML code), data extraction for commercial purposes and feed fetching Web content. To perform these actions, bots visit Web servers repeatedly and, in some cases, for a prolonged period of time [10].

However, allowing bots unrestricted access to Web server content and services is not a good practice. This is because bots are a powerful tool that has often been abused for malicious purposes, such as Web content scraping, vulnerability scanning, marketing fraud, carding, account takeovers, spamming, denial of service attacks and more [10]. Furthermore, it is possible to operate bots from mobile phones and IoT devices (usually without the device owner's knowledge or consent) which makes them a low cost mechanism for distributed attacks [10]. Moreover, advanced Web bots can also avoid detection by imitating humanlike behavior [10]. As a result, in addition to being a crucial part of the infrastructure of the Internet, Web bots have become a ubiquitous threat.

The need to mitigate this threat has inspired a whole new area of research. Once a visitor is identified as a Web bot the process is straightforward – websites simply need to place special restrictions, so that bots cannot perform malicious acts. To detect Web bots, current state of the art approaches both in academia [13, 15, 31] and in commercial solutions [1, 10] propose, besides the rule-based Web bot detection techniques, the use of machine learning to distinguish bots from human visitors. Such approaches rely on the collection of Web server logs that contain both human and Web bot sessions. These data can be used for modelling human and bot activity.

Current research on machine learning based Web bot detection performs automatic annotation of the extracted sessions from Web logs to create the ground truth needed to generate such models [25, 26]. Such approaches do not consider that the bots may try to hide their bot nature and/or try to imitate humanlike behaviour [10]. Thus, we performed a more in depth analysis of the Web bot detection problem by examining the detection accuracy of machine learning based algorithms in terms of identifying conspicuous Web bots separately from its performance in detecting advanced Web bots which try to hide their bot nature.

We created a Web bot detection framework which examines HTTP Web logs from a public Web server. The log data were split in three categories: (i) simple Web bots which do not try to hide their bot nature, (ii) advanced Web bots which use a browser fingerprint and may present a humanlike browsing behaviour and (iii) human sessions which we assume to be all the sessions that do not belong to the above categories. Furthermore, differentiating from relevant literature, we studied the behaviour of our framework in a false-positive intolerant Web server. We opted to do this because in a real-world case scenario miscategorising human visitors is not desirable since it will affect the browsing experience of visitors.

The main contributions of this paper are:

- The proposal of a modular machine learning based Web bot detection framework that (i) can be easily combined with any HTTP Web server and (ii) can effortlessly incorporate new machine learning-based Web bot detection algorithms.
- The identification of the unique challenges when state-of-the-art Web bot detection techniques are utilised for detecting advanced bots as opposed to simple bots
- The identification of the most important features among the ones proposed in literature for the detection of simple and advanced Web bots.

The remainder of this paper is structured as follows: Section 2 provides the background on the Web bot detection problem and covers the related work. Section 3 describes the Web bot detection framework. Section 4 presents our evaluation methodology and experimental setup and Section 5 contains the evaluation results. Finally, Section 6 discusses the significance of our results to the Web bot detection problem and Section 7 summarizes our work and examines the future evolution of this framework.

2 BACKGROUND AND RELATED WORK

The Web bot detection problem poses the question of how we can accurately distinguish whether a Web visitor is a bot or a human. Researchers have further categorised bots based on their functionality [12] or purpose (benign/malicious) [4, 23, 31].

In the past, to detect Web bots, it used to be common to examine the signature of the visitor's request, i.e. the request headers, and whether JavaScript, cookies, and Web sessions are supported. However, tools, such as the selenium[1], provide APIs that allow bots to mimic the signature and support the majority of the features of most common browsers, including the support of JavaScript, cookies and sessions.

Currently, the most famous techniques for Web bot detection are based on the CAPTCHA (i.e. Completely Automated Public Turing test to tell Computers and Humans Apart) [28] such as the reCAPTCHA[2] offered by Google. The CAPTCHA is a Turing test that is based on a visual challenge, accompanied with an aural one for the visually impaired. The test uses the assumption that a human can extract letters from either a distorted image or the audio file or select an object in an image, while a Web bot cannot.

However, a variety of techniques have been proposed to bypass some of the most popular CAPTCHA challenges, such as the use of public online speech to text engines to bypass Google's reCAPTCHA [6]. Finally, the CAPTCHA has received a lot of criticism, especially from people with disabilities who sometimes struggle with fulfilling this request and people who feel that their everyday work is slowed down.

To solve the aforementioned problems, current research focuses on the use of machine learning based detection techniques to distinguish Web bots from humans, rather than solely relying on rule-based detection techniques. The first step in generating machine learning models that can be used for the detection of Web bots is the extraction of sessions from Web logs [5, 18, 22, 24, 25]. After that, several features are extracted from each session and used to identify whether the visitors are bots or humans. These features include the access frequency of Web pages [27], the type of Web content accessed (i.e. HTML, text, JavaScript, image, css, etc.) [13, 22], the access patterns [3] and the HTTP errors produced [5, 27]. These features are used as input to generate machine learning models.

The most popular machine learning based Web bot detection problems that appear in research are the classification [25, 26] and clustering [2, 9, 27]. The detection can take place either off-line (i.e. decide after the end of the sessions whether it is from a bot), or online by performing an estimation during the session [8, 21]. In both cases, a ground truth of human and Web bot sessions is required. In most recent research, the annotation process relies on comparing each visitor's agent name [26] and IP address [5, 7, 13, 22, 25] with the agent names and IPs of known web bots according to lists hosted on external servers. Such lists mostly contain identifiers for bots which are benign in nature, like, for example, search engine bots, although some malicious bots can be found there as well. Furthermore, some researchers examine whether the visitors access a text file which instructs Web robots which Web pages to crawl/scrape[3] [15, 27, 31].

Although the aforementioned techniques show promising results, they do not address a key aspect of the Web bot detection problem, which is the identification of Web bots which try to evade detection via, for example, presenting a humanlike fingerprint and, in some cases, behaviour. More specifically, such bots can simply

[1]http://www.seleniumhq.org/
[2]http://www.google.com/recaptcha
[3]The robots text file (robots.txt) that is located in the root folder of a Web server.

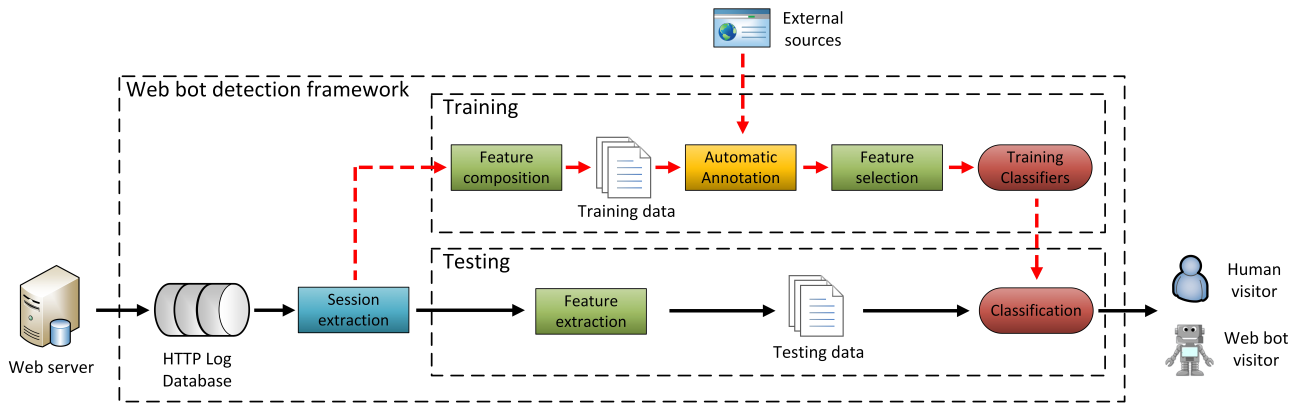

Figure 1: The Web bot detection framework

use a browser-like agent name, randomize their IPs and not access the robots text file. The techniques that have been proposed by literature mislabel these bots. Even worse, in the less likely scenario where an advanced Web bot presents human-like behaviour, it could be mislabeled even from human annotators.

Therefore Web bot detection needs to be treated as a multifaceted problem. To this end, we isolated the advanced Web bots so as to accurately assess the performance of our detection framework in terms of detecting advanced Web bots separately from simple Web bots. To achieve that, we propose a novel annotation technique which combines the list of known Web bot agent names (proposed in literature) with an external honeypot to examine whether the visitor's IP has shown activity on that honeypot. Specifically, instead of checking whether the visitor's IP is a known Web bot's IP, we check whether this IP has shown activity on a honeypot server. The interaction with the honeypot indicates bot activity, since no human user would have visited that server. We made this choice because several bots in our dataset were already labeled as bots in the honeypot, but were not yet in the list of known Web bots.

3 THE WEB BOT DETECTION FRAMEWORK

In this section we present the Web bot detection framework that we created to examine how well common Web bot detection algorithms perform in detecting simple and advanced Web bots under various configurations. This framework is based on and combines the most prevalent techniques that have been proposed in literature for Web bot detection using supervised machine learning [25, 26].

The architecture of the Web bot detection framework is shown in Figure 1. The input of the framework is a directory path in which the HTTP logs from the Web server are stored. The framework uses a regular expression to extract the relevant content from HTTP logs. Thus, the process of applying different log files as input is trivial, since any new format of interest can be incorporated by only adapting this regular expression rule.

After the successful connection of the framework with the HTTP server log files, the session extraction procedure takes place, where HTTP log data are split into sessions (Section 3.1). For each session, a feature vector is created using a set of features proposed in literature (Section 3.2). After that, each session is annotated as Web bot or human using an automated way (Section 3.3). Furthermore, the importance and effectiveness of each feature is evaluated and a subset is selected (Section 3.4). Finally, the selected feature vectors are used to create the classification models (Section 3.5).

In the testing phase, the previously created classification models are used to identify Web bot sessions in new unseen data. When new data are available, their sessions and features are extracted accordingly (Sections 3.1 and 3.2). Each classifier uses a unique subset of the available features which consists of the ones that were deemed most important during their training stage (Section 3.4). The trained classifiers take the new data as input and determine whether each visitor is a bot or a human (Section 3.5).

3.1 Session extraction

The first step in identifying whether a visitor is a human or a Web bot is the extraction of the visitor's session(s) from the log files. Since several visitors (including bots and humans) might share the same IP, considering only the IP field to extract sessions initiated from different visitors is insufficient. For this reason, current research proposes the combination of the IP with the browser agent name for the creation of a unique identifier per visitor [5, 13, 15, 22, 25].

The aforementioned technique will not necessarily result in distinguishing all users from each other, since there might be two users with the same IP and agent name or one user changing several agent names in rotating order. To this end, relative research proposed more advanced fingerprinting techniques, such as browser-based characteristics (e.g. ActiveX support, Flash enabled, language enumeration, etc.), OS and applications features (e.g. OS and kernel version, Windows registry, etc.) and hardware features (e.g. screen resolution) [19]. However, since this information was not available in the log data that we used, we followed the default approach of identifying separate sessions by the combination of a unique IP - agent name pair [5, 13, 15, 22, 25].

To define when a user session has ended, current research uses a 30 min threshold [5, 13, 15, 22, 25]. More specifically, when a session id stays idle for more than 30 minutes, a new session is created upon a new request. Furthermore, sessions that have a total number of HTTP requests lower than a threshold k were not taken

Table 1: Attributes calculated for each session.

Id	Feature	Short description	Literature
1	Total requests	Total number of HTTP requests that the agent issued during the session.	[2, 11, 25–27, 31]
2	Total session Bytes	The sum of all requested pages' size (in Bytes) in a session.	[2, 7, 21, 27, 31]
3	HTTP GET requests	Total number of HTTP GET requests issued during the session.	[3, 4, 7, 25, 31]
4	HTTP POST requests	Total number of HTTP POST requests issued during the session.	[4, 7, 25, 31]
5	HTTP HEAD requests	Total number of HTTP HEAD requests issued during the session.	[4, 7, 11, 21, 25–27, 31]
6	% HTTP 3xx requests	The percentage of HTTP requests that led to an HTTP 3xx code response.	[3, 7, 31]
7	% HTTP 4xx requests	The percentage of HTTP requests that led to an HTTP 4xx code response.	[3, 7, 11, 21, 26, 27, 31]
8	% image requests	The percentage of HTTP requests that requested an image. This feature searches for all known image formats' ending.	[13, 22, 25]
9	% pdf requests	The percentage of HTTP requests that requested a pdf file.	[11, 13, 26, 27]
10	% css file requests	The percentage of HTTP requests that requested a css file.	[22]
11	% js requests	The percentage of HTTP requests that requested a JavaScript file.	[13, 22]
12	HTML-to-image ratio	The number of the requested HTML files divided by the number of requested image files in a session.	[11, 26, 31]
13	% requests with unsigned referrers	The percentage of total HTTP requests that had no refer.	[4, 7, 11, 25–27]
14	Search engine refer	Binary. If a session has at least one request with a known search engine refer.	[4]
15	Unknown refer	Binary. Refer exists, but not from the aforementioned search engines.	[4]
16	Depth SD	Standard deviation of requested pages' depth (i.e. number of '/' in URL path).	[26, 27, 31]
17	Max requests per page	The maximum number of requests to the same page in a session.	
18	Average requests per page	The average number of requests per page in a session.	
19	Max number of consecutive sequential HTTP requests	The maximum number of HTTP requested URLs that contain the previously requested URL as a subpart page	[31]
20	% of consecutive sequential HTTP requests	The percentage of HTTP requested URLs that contain the previously requested URL as a subpart.	[11, 26, 27]
21	Session time	The total time (in seconds) between the first and the last HTTP request of the session.	[2, 3, 21, 25, 31]
22	Browsing speed	The ratio of the total number of requested pages over time (in seconds).	[3]
23	SD of inter-request times	Standard deviation of time between successive requests.	[3]

into consideration because it is not feasible to distinguish bots and humans based on only a few HTTP requests [22].

3.2 Feature composition

The information included in each session is encoded into measurable values and used as input to train the classification models. To decide which measurable "properties" or "characteristics" (features) to consider, we accumulated the most promising features that have been proposed over the past 5 years. These features are presented in Table 1, along with a short description for each feature and relevant literature. In short, to distinguish Web bots from humans we can examine the method and response code of the HTTP request, the type of file(s) requested and the browsing behaviour.

3.3 Automatic annotation

The extracted sessions that are used for training the classifiers are annotated as "bot visitor sessions" or as "human visitor sessions". Bot visitor sessions contain two different types of sessions; (i) those in which the Web bots are conspicuous, i.e. they are not trying to hide the fact that they are bots, and (ii) those in which the Web bots are inconspicuous, i.e. they replace one or more of their normal bot characteristics with those of a human visitor to remain undetected.

The annotation process which we followed is depicted in Figure 2 and is a two step process. The first step is to identify simple Web bots by examining whether their agent name is a browser one, while the second one is to identify, to the best of our ability, Web bots that present a browser fingerprint and, in some cases, a humanlike behaviour by using an external honeypot. Initially, we used the API provided by Useragentstring[4], a server that classifies agent names in several categories such as "browser", "crawler", "library", "link checker". After that, we used the API provided by the GreyNoise[5], a server that collects and analyzes untargeted, widespread, and opportunistic scans and attacks or malicious activities, to check whether the IPs have been found to perform any of the above.

The main idea behind our approach is that it is not common for a human visitor to change the agent name of their browser. Thus, if a session has a non-browser agent name, it can be safely annotated as Web bot. However, all sessions that have a browser agent name

[4]http://useragentstring.com/
[5]https://greynoise.io/

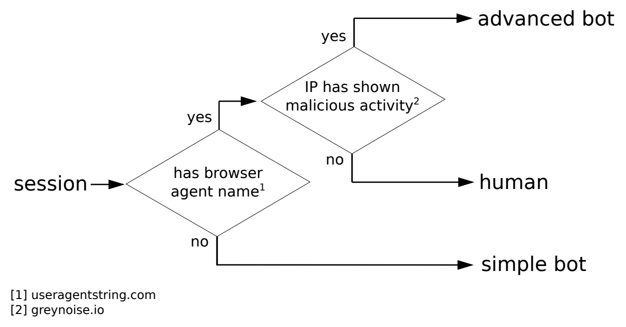

Figure 2: Automatic annotation process

are not necessarily from human visitors, since bots might change their agent name to present a browser-like fingerprint [10, 16, 30].

3.4 Feature analysis and selection

Feature selection is described as a method whereby specific features are selected from the set of all available features. In machine learning, and more specifically, classification problems, some features might result in the decrease of the models’ accuracy. For this reason, feature selection is widely used in machine learning based problems and has also been used in security related tasks that use machine learning techniques [9].

The framework supports the analysis of all the available features and the selection of the most important ones. Two selection modes are supported, one which selects the most promising features based on the feature analysis regardless of the classification algorithm employed and one that is classifier dependent. In both cases, each classifier is accompanied by a boolean array that indicates which features are to be used from all the available features for its training. The same features are used in the testing process.

3.5 Classification

The classification process consists of two phases, the training phase and the testing phase. In the training phase we feed the framework with known Web bot and human sessions as input to create the classification models. These models are stored into the framework so that they can be used for the testing process.

The testing process is similar to the training one. The new (unseen by the classifiers) Web sessions are used as input to the framework and the classifiers generate the respective label (Web bot or human). To evaluate the framework, the labels of the sessions used for testing are known (but kept secret). However, in a real case scenario, the nature of these sessions would be unknown.

4 EVALUATION

To assess the effectiveness of the proposed framework in terms of identifying Web bot sessions from HTTP log files, a series of experiments were conducted using real HTTP traffic collected from a public Web server. This section describes the evaluation methodology (Section 4.1), the dataset (Section 4.2), the feature analysis and selection process (Section 4.3), the evaluation metrics considered (Section 4.4) and, finally, the classification algorithms tested and their configuration (Section 4.5).

4.1 Evaluation methodology

The purpose of this paper is to identify the unique challenges that arise when state-of-the-art Web bot detection techniques are utilised for detecting advanced Web bots as opposed to simple bots. To this end, we evaluated our framework in how well it can identify simple and advanced Web bots separately. Initially, we identified the most important features in the case of simple and advanced Web bots. We used these features to generate the respective classification models and evaluate our framework after the models’ deployment.

Furthermore, we took into account the fact that, in a real world case scenario, it is imperative to have a low false positive rate in order to avoid miscategorising human visitors. Thus, we tested our framework’s general performance in various working points (i.e. classification thresholds) and analysed its performance on the working point in which the false positive rate is relatively low.

4.2 Dataset

The framework was tested on HTTP log data collected from MK-Lab’s public Web server[6]. Instead of feeding the framework with data real time, we used a year’s worth of HTTP log data (from 20/3/2016 to 20/3/2017) as input in a simulated time mode. We only considered Web sessions with more than k=30 requests per session to ensure that the framework has adequate data to identify the nature of each visitor [22]. The value of k was chosen heuristically.

We annotated the dataset by examining the agent name of the visitor as well as whether its IP has shown malicious activity (see Section 3.3). The total unique agent names extracted from sessions with more than k=30 requests were 2793. From them, the 2723 were annotated by the useragentstring’s API as "browser" (2628) or "bot" (95) and 70 were annotated as "unknown". For the "unknown" agent names, we manually annotated them as browsers (66) or bots (4).

As we mentioned in Section 3.3, the IPs of the sessions that were annotated as "browsers" by the useragentstring’s API (15452) are also checked for malicious activity using the GreyNoise’s API. Thus, we end up changing the annotation of 299 unique IPs (554 sessions) which were originally annotated as "browser" but their IPs have been marked as bots by the GreyNoise. The total unique agent names and IPs per class (i.e. browser, simple bot, advanced bot) are shown in Table 2.

Table 2: Unique agent names and IPs

	Bots			Humans	Total
	Simple	*Adv.*	*Total*		
Agent names	99	105	204	2589	2793
IPs	602	299	901	15153	16054

To evaluate the framework, we split the dataset into two sets, one for training and one for testing. Our Web server by default splits the HTTP log data into files based on a log rotation technique[7]. The total files that were generated over a year were 13. We grouped the files into two packages, (i) the training one using the first 8 files (from 20/3/2016 to 4/12/2016) and (ii) the testing one using the

[6]Multimedia Knowledge and Social Media Analytics Laboratory, https://mklab.iti.gr/
[7]https://httpd.apache.org/docs/2.4/logs.html

other 3 files (from 4/12/2016 to 20/3/2017). For each of these files we extract all sessions with more than k=30 requests per session [22].

The final number of extracted sessions is shown in Table 3. To assess the framework's performance in identifying simple bots and advanced bots separately, we created two "sub-datasets", the D1 that contains the human and simple bot sessions and the D2 that contains the human and the advanced Web bot sessions.

Table 3: Human and Web bot sessions

	Bots			**Humans**	**Total D1**	**Total D2**
	Simple	*Adv.*	*Total*			
Train	1321	431	1752	17462	18783	17893
Test	1034	123	1157	6195	7229	6318
Total	2355	554	2909	23657	26012	24211

4.3 Feature analysis and selection

To analyse the importance of the extracted features in the detection of simple as well as advanced Web bots we utilised the Principal Component Analysis (PCA) and the x^2 (chi-square) feature selection techniques. For both of these techniques the data were scaled. When scaling the data for the PCA, we subtracted the mean values and then divided by standard deviation for each feature in the training set. In the case of x^2, we divided by standard deviation without subtracting the mean to avoid negative values. We then used the mean and standard deviation values calculated from the training set to perform the same process in the testing set.

PCA can be used to assess the importance of each feature by calculating its contribution to the generated principal components. To do that, we can measure the mean of each feature "contribution" to all the generated components of the PCA using the training set [17] (D1 for simple bots and D2 for advanced bots - see Section 4.2). Usually, the smaller principal components (i.e. with lower variance) are associated with noise and thus they can be omitted. Thus, the features with the lowest cumulative "contribution" to all the principal components can also be associated with noise [17]. However, the high variance principal components are not necessarily all useful, since they might not be correlated with the respective class (i.e. Web bot or human) or they may refer to noise existing within the data. Thus, we combined the results of the PCA technique with the x^2 feature selection technique to see the most important features to the Web bot detection problem.

To select the features that will give us the highest score for each classifier, we used the greedy Sequential Feature Selection (SFS) technique [20]. The SFS works as a wrapper on top of each classifier. It is an iterative process where in each iteration the feature with the highest metric (in our case balanced accuracy) on the training set is chosen and added to the features that are used for each classifier. By this way, the features that perform the worst will be added in the end and can omitted if they do not contribute to the performance. Thus, SFS provides different results for each classifier, which is useful because each feature may contribute differently depending on the classifier that was used.

4.4 Evaluation metrics

Several researchers used accuracy as the evaluation metric for Web bot detection [21, 22, 26, 29]. Since it is possible for an algorithm to have high accuracy while maintaining low precision, other researchers use the precision and recall metrics as well to assess the performance of the proposed approaches more accurately [2, 11, 13, 21, 25, 26]. Furthermore, researchers also calculated the harmonic mean of the precision and the recall which is called F-measure, F1 score or simply F-score [11, 13, 21, 25, 26].

Due to the unbalanced classes in our dataset, we decided to use balanced accuracy as opposed to accuracy. Furthermore, to evaluate the framework's performance in both the case of Web bot detection as well as human user detection we calculated the precision, recall and their harmonic mean, F-score, for both classes. Finally, to gain a more general understanding of the performance of the classifiers irrespective of the working point (i.e. classification threshold) we considered the Area Under Curve (AUC) evaluation metric that can be calculated by plotting the Receiver Operating Characteristic (ROC) curve for a classifier [14].

4.5 Classification algorithms tested

Our framework is built to allow for the effortless incorporation of any machine learning algorithm. For our experiments, we incorporated 4 well known machine learning algorithms, all of whom have been used by other researchers for the Web bot detection problem. More specifically, we incorporated the Support Vector Machine [21, 26], the Random Forest [25], the Adaboost [25] and the MLP classifiers [7, 21, 26]. Furthermore, we added an ensemble classifier, which we call the Voting classifier, that performs a class probability averaging of all the available classifiers [25].

The parameters for each classifier are shown in Table 4. We performing an exhaustive search over specified parameter values for each classifier and chose the ones which have the highest balanced accuracy with a 2-fold cross validation on the training data. Furthermore, in the case of SVM and MLP Classifier, the data are scaled to avoid the problem of domination of some features over the others. To scale the data, we followed the same scaling technique that we used in the PCA (Section 4.3)

For the implementation of these algorithms the scikit-learn[8] Python library was used. Furthermore, all the experiments were performed on an Intel processor at 3.4GHz and 32GB RAM for loading large datasets during the experiments.

5 RESULTS

In this section we present the results of the evaluation of our framework in regards to detecting simple and advanced Web bots. First, we analyse and select the most important features for each classification algorithm (Section 5.1). Subsequently, we evaluate the general performance of our framework (Section 5.2) and the performance of our framework in a false positive intolerant server (Section 5.3).

[8]http://scikit-learn.org/stable/

Table 4: The parameters used on the classification algorithms.

Classification Algorithm	D1 - Simple Web bots	D2 - Advanced Web bots
SVC	RBF kernel, C=16384, gamma = 1.22 * 1e-4, tol=0.001	RBF kernel, C=64, gamma=2, tol=0.001
MLP Classifier	tanh activation, adam solver, a=0.1, b1=0.9, b2=0.9, e=1e-05, hidden layer sizes: (100, 50), constant learning rate	tanh activation, sgd solver, a=1, b1=0.1, b2=0.1, e=1e-08, hidden layer sizes: (400), invscaling learning rate
Random Forest	Estimators = 200, Gini criterion, Max features =$\sqrt{no\_features}$, Min samples per leaf = 4, min samples split 10, max depth=70	Estimators = 1000, Gini criterion, Max features =$\sqrt{no\_features}$, Min samples per leaf = 4, min samples split 2, max depth=10, out-of-bag samples used
Adaboost	Decision Tree Classifier as base estimator, estimators=450, decision entropy criterion, no max depth, Max features =$\sqrt{no\_features}$, "best" split strategy, learning rate=1	Decision Tree Classifier as base estimator, estimators=50, decision entropy criterion, no max depth, Max features =$\sqrt{no\_features}$, "best" split strategy, learning rate=1

5.1 Feature analysis and selection

5.1.1 Feature analysis. Figure 3 presents the cumulative variance of the data by adding the PCA's principal components one at a time ordered by descending eigenvalues for the simple (D1) as well as the advanced (D2) Web bots. As we can see, the principal components generated from D1 and D2 have similar variance. Moreover, the number of principal components which are essential to maintain at least 90% and 95% of the variance for simple and advanced Web bots is 14 and 16, respectively.

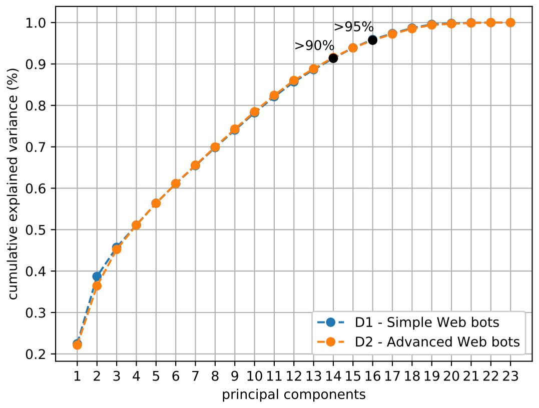

Figure 3: Cumulative variance of PCA's components for simple (D1) and advanced (D2) Web bots

Furthermore, we calculated the absolute value of the mean values of each feature "contribution" to all the components for the simple (D1) and advanced (D2) Web bots (Figures 4). In general, the higher the mean value of the features' contribution to the principal components the more important the feature can be considered. However, this is not always the case; some of these features might not contribute to the problem. Thus, to decide which features are the most important, we also calculated the respective x^2 scores for the simple (D1) as well as the advanced (D2) Web bots (Table 5).

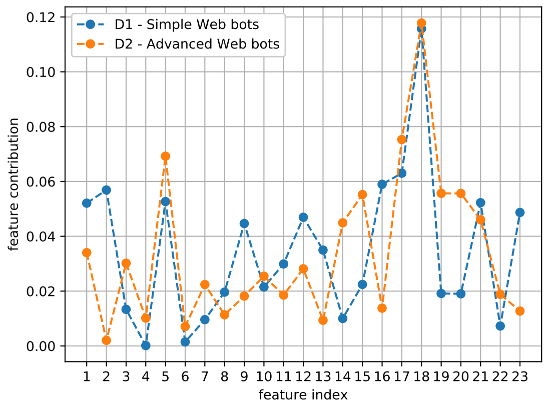

Figure 4: The absolute mean value of each feature contribution to PCA components for the simple (D1) and advanced (D2) Web bots

Table 5: Ranked features using x^2 for simple (D1) and advanced (D2) Web bots

Dataset	Ranked features using x^2
D1 - Simple Web bots	13, 21, 23, 9, 15, 5, 10, 11, 8, 14, 7, 22, 16, 4, 6, 17, 12, 18, 19, 3, 20, 2, 1
D2 - Advanced Web bots	17, 4, 21, 12, 18, 1, 6, 3, 22, 23, 8, 9, 16, 11, 7, 13, 10, 5, 14, 2, 20, 15, 19

In both cases (PCA and x^2), the importance of the features differs when using the D1 dataset (simple Web bots) and when using the D2 (advanced Web bots). Furthermore, the features that have both

Table 6: Ranked and selected (bold and in brackets) features using SFS for simple (D1) and advanced (D2) Web bots for each classification algorithm

Classification Algorithm	D1 - Simple Web bots	D2 - Advanced Web bots
SVM	**{13, 4, 20, 8, 22, 14, 18, 21, 16, 1, 11, 19, 10, 17, 23, 5, 9, 12, 3, 6, 2, 7, 15}**	**{17, 9, 20, 7}**, 5, 2, 14, 22, 6, 4, 11, 10, 8, 23, 3, 1, 16, 12, 19, 13, 15, 21, 18
MLP Classifier	**{13, 15, 4, 14, 23, 20, 18, 5, 2, 21, 17, 12, 7, 6, 22, 8, 9, 1, 3, 10, 16, 11}**, 19	**{17, 21, 23, 6, 12, 9, 19, 5, 22, 14, 18, 16, 20, 10, 1, 8, 11, 15, 2, 3, 7, 13, 4}**
Random Forest	**{13, 4, 11, 19, 14, 20, 6, 17, 23, 5, 9, 12, 10, 8, 21, 18, 1, 16}**, 22, 7, 2, 15, 3	**{11, 14, 12, 20, 10, 5, 21, 7}**, 9, 6, 8, 4, 2, 17, 22, 23, 1, 15, 3, 16, 18, 19, 13
Adaboost	**{13, 4, 19, 14, 20, 15, 5, 9, 10, 8, 11, 7, 17, 18, 12, 3, 6, 16, 1}**, 21, 23, 2, 22	**{11, 8, 14, 20, 10, 5, 9}**, 2, 22, 7, 6, 4, 12, 1, 3, 17, 18, 21, 23, 19, 15, 13, 16

the higher contribution to the PCA and a high x^2 score are 21 and 23 for simple Web bots (D1) and 17 and 18 for advanced Web bots (D2). Features 21 and 23 have to do with time-related aspects of the browsing behaviour of the visitor. Simple Web bots, usually, have a predefined and therefore predictable behaviour regarding time, which is why they can be detected this way. On the other hand, advanced Web bots use a more unpredictable browsing behaviour, so the characteristic that gives them away is the unique content they try to access (Features 17 and 18).

5.1.2 Feature selection. Each feature might contribute differently in the performance of different classification algorithms. Thus, we performed the greedy SFS technique to pick the features that give the highest score (in our case balanced accuracy) for each classifier in the case of simple (D1) as well as advanced (D2) Web bots. We decided to keep as many features as possible as long as they do not noticeably decrease the balanced accuracy in training. The selected features for each classifier and dataset are shown in Table 6.

The SFS results show that each feature contributes differently in each classifier. Furthermore, the initial features selected by the PCA in combination with the x^2 were selected and highly ranked in some classifiers and rejected in other classifiers. Such an example is feature 17, which was initially selected and is highly ranked in the case of SVM and MLP Classifier, but rejected in the case of the Random Forest and Adaboost.

For this reason, depending on the size of the dataset and the processing power we have, we can either select the most promising features according to the combination of the PCA with the x^2 technique or perform the greedy SFS over all the features and pick the ones that perform better on the training set. In our case, since the dataset was relatively small, we followed the latter.

5.2 General Performance

To evaluate the general performance of our framework, we plotted the ROC curve of the Voting classifier when the framework was tested on simple and advanced Web bots (Figure 5). We also marked a few working points (i.e. classification thresholds) based on the respective False Positive Rate (FPR). We opted to do this because in a real-world scenario a Web bot detection framework must be false-positive intolerant to avoid affecting visitors' experience.

The performance of our classifiers show that detecting simple Web bots is a trivial task. The framework is able to effectively

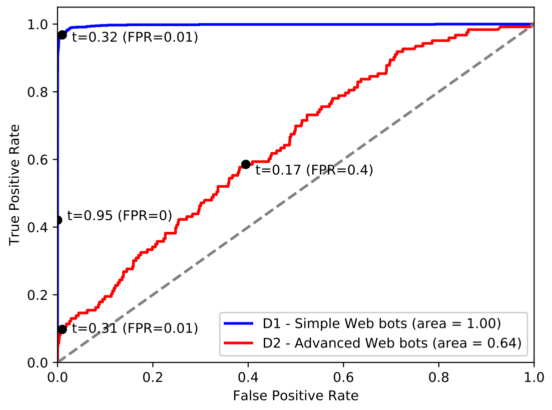

Figure 5: ROC curve of the Voting classifier for the simple (D1) and advanced (D2) Web bots

detect the simple Web bots (D1 dataset) with an AUC=1.00. However, detecting advanced Web bots (D2 dataset) is not that simple. The framework performs poorly and, if a low FPR is required, the framework detects very few Web bots.

To further analyse the behaviour of our framework on the selected working points in the case of advanced Web bots (D2), we calculated the confusion matrix of the Voting classifier on the two working points selected in Figure 5 (Tables 7 and 8).

Table 7: Confusion matrix for advanced Web bots (FPR=0.4, t=0.17)

		Predicted Values		
		Bot	Human	Total
Actual Values	Bot	82	41	123
	Human	3661	2534	6195
	Total	3743	2575	

The choice of a working point depends on how strict we want our detection framework to be in each case. For example, choosing a working point with FPR = 0.4, we would correctly identify 2 out of 3 advanced Web bots, but most humans would be misclassified

Table 8: Confusion matrix for advanced Web bots (FPR=0.01, t=0.31)

		Predicted Values		
		Bot	Human	Total
Actual Values	Bot	18	105	123
	Human	417	5778	6195
	Total	435	5883	

(Table 7). Choosing a higher threshold (lower FPR) results in fewer misclassified human visitors, but the framework's effectiveness in detecting advanced Web bots is decreased.

5.3 Performance on a false positive intolerant Web server

To assess the framework's performance on a false-positive intolerant Web server, we calculated the precision, recall and F-measure in the working point of FPR=0.01 for all employed classifiers. Furthermore, we calculated the balanced accuracy, which represents more accurately the performance of the framework in the case of unbalanced datasets. The results are shown in Figure 6.

The performance of the classifiers shows that identifying advanced Web bots is more challenging than identifying simple Web bots. When choosing a working point with a low FPR, simple bots are detected with very high precision (~95%) and recall (~97%) which makes an F-measure higher than 96%. Furthermore, in the case of detecting human visitors (class 0), we achieved a precision and recall of more than 99% each. However, the framework achieves low precision and recall in the case of advanced Web bots which results in a low balanced accuracy (~55%).

Moreover, the performance of different classifiers varies. To achieve a more balanced behaviour we chose the Voting classifier to be the main classifier. Generally, voting classifiers are not always guaranteed to have a better performance. However, they can be more "stable", since, if one of the employed classifiers underperforms, its behaviour will be masked by the other classifiers. For example, Random Forest achieves the highest balanced accuracy in the case of advanced Web bots (D2) but, at the same time, very low recall of the human class (Figure 6).

6 DISCUSSION

There is a huge incentive for individuals and companies alike to create Web bots that can bypass Web bot detection techniques. This has led to the introduction of advanced Web bots that try to evade detection. Our dataset, which is comprised by the logs from a public web server, contains several sessions from such bots. We used these logs to determine the effectiveness of state-of-the-art Web bot detection techniques against advanced web bots. The results have shown that, although detecting simple bots is relatively easy, detecting advanced Web bots that present a browser fingerprint and maybe a humanlike behaviour is much more difficult. Furthermore, if we try to detect such bots with current detection techniques, we will end up misclassifying benign visitors, which is a non-desirable behaviour in a real-world case scenario.

Literature has focused on identifying all kinds of Web bots, treating Web bots as one group of visitors. However, since advanced Web bots will be considerably fewer than simple bots, the aforementioned technique will present biased results masking its ability or lack thereof to detect advanced bots. To this end, we performed a more in depth study of the Web bot detection problem by dividing the simple from the advanced Web bots and showed that there is an efficiency gap in the detection of advanced web bots compared to simple ones. For this reason, we have concluded that the features proposed by literature were not suitable for the detection of advanced bots.Future work includes examining browsing behaviour holistically instead of relying exclusively on requested pages and time. For example, we could use features extracted from visitor mouse movements and keystrokes. By incorporating such features in our framework we will be able to more accurately identify advanced Web bots.

In summary, the question of whether existing detection mechanisms can be used as an effective solution comes down to the threat model. If we choose to only target simple Web bots (which is the majority of bots that will visit our Website) we can easily detect them using using hard-coded rules. Even if such bots try to evade detection by presenting a browser fingerprint, they can easily be detected by their behaviour by using machine learning models from features extracted from HTTP logs. However, if we are targeting advanced Web bots, we need to have a better understanding of their behaviour and use more advanced features generated from more sources than simply the commonly used by research HTTP logs.

7 CONCLUSIONS

This work presented an in depth analysis of the Web bot detection problem by examining the performance of a machine learning based Web bot detection framework in terms of identifying simple Web bots separately from its performance in detecting advanced Web bots which try to hide their bot nature. To do that, we generated the ground truth to train our models by using a novel automatic annotation mechanism that examines (i) the fingerprint of the visitor (in our case the agent name) as well as (ii) whether its IP has shown malicious activity using an external honeypot.

The proposed framework was tested on real HTTP Web log data collected from a public Web server. The results of our evaluation experiments indicated that the Web bot detection problem is a multifaceted one, characterised by the coexistence of simple bots that can be detected easily and advanced Web bots that are considerably more difficult to detect. Furthermore, if the framework is applied on a false-positive intolerant Web server, its effectiveness regarding detecting advanced Web bots is significantly reduced.

Future work includes the introduction of more advanced features that can not be easily simulated by bots to facilitate the identification of advanced Web bots. These features will be aggregated to our framework. Furthermore, we are planning to examine and improve the framework's performance in adversarial settings, where adversaries try to create bots that adjust their behaviour dynamically to avoid detection.

ACKNOWLEDGMENTS

This work was supported by the TENSOR (H2020-700024) and Ideal-Cities (H2020-778229) projects, funded by the European Commission.

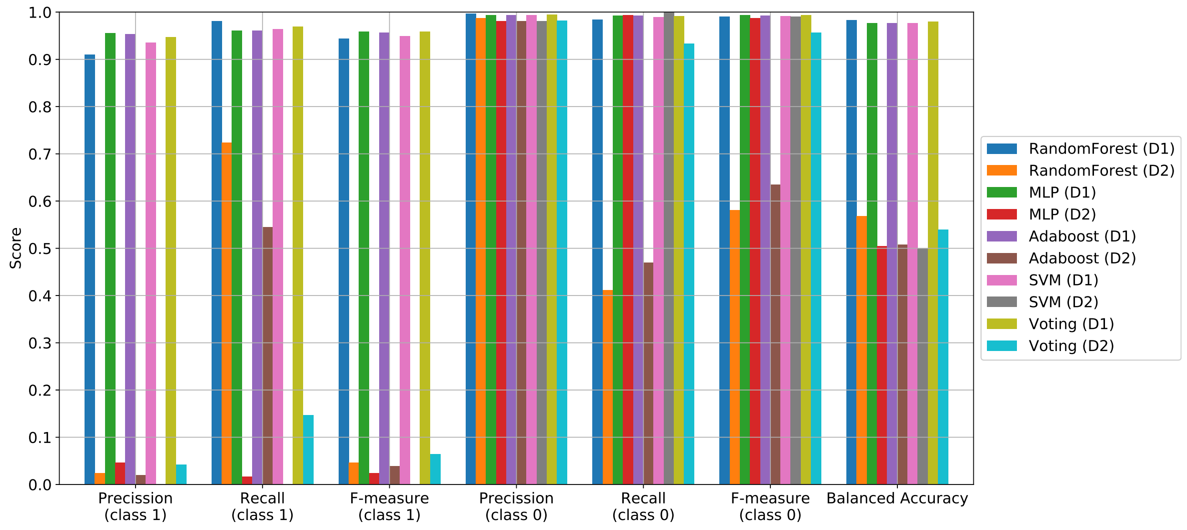

Figure 6: Comparison of the effectiveness of the classification algorithms for humans (class 0) and Web bots (class 1) for the D1 (simple Web bots) and D2 (advanced Web bots) datasets in the working point with FPR=0.01

REFERENCES

[1] Akamai. 2018. 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

[2] Shafiq Alam, Gillian Dobbie, Yun Sing Koh, and Patricia Riddle. 2014. Web bots detection using particle swarm optimization based clustering. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2955–2962.

[3] Yasmin A AlNoamany, Michele C Weigle, and Michael L Nelson. 2013. Access patterns for robots and humans in web archives. In *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 339–348.

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

[5] Anshul Bhargav and Munish Bhargav. 2014. Pattern discovery and users classification through web usage mining. In *Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on*. IEEE, 632–636.

[6] 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} 17)*.

[7] Alberto Cabri, Grażyna Suchacka, Stefano Rovetta, and Francesco Masulli. 2018. Online Web Bot Detection Using a Sequential Classification Approach. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 1536–1540.

[8] Zi Chu, Steven Gianvecchio, and Haining Wang. 2018. Bot or Human? A Behavior-Based Online Bot Detection System. In *From Database to Cyber Security*. Springer, 432–449.

[9] Zibusiso Dewa and Leandros A Maglaras. 2016. Data mining and intrusion detection systems. *vol* 7 (2016), 62–71.

[10] Distil Networks. 2018. 2018 BAD BOT REPORT: The Year Bad Bots Went Mainstream. https://resources.distilnetworks.com/white-paper-reports/2018-bad-bot-report

[11] Wang Dong, Xi Lei, Zhang Hui, Liu Hebing, Zhang Hao, and Song Ting. 2015. Web robot detection with semi-supervised learning method. (2015).

[12] Derek Doran and Swapna S Gokhale. 2012. A classification framework for web robots. *Journal of the Association for Information Science and Technology* 63, 12 (2012), 2549–2554.

[13] Derek Doran and Swapna S Gokhale. 2016. An integrated method for real time and offline web robot detection. *Expert Systems* 33, 6 (2016), 592–606.

[14] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[15] Javad Hamidzadeh, Mahdieh Zabihimayvan, and Reza Sadeghi. 2017. Detection of Web site visitors based on fuzzy rough sets. *Soft Computing* (2017), 1–14.

[16] Gregoire Jacob, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2012. PUBCRAWL: Protecting Users and Businesses from CRAWLers.. In *USENIX Security Symposium*. 507–522.

[17] Otterbach Johannes. 2016. Principal Component Analysis (PCA) for Feature Selection and some of its Pitfalls. http://jotterbach.github.io/2016/03/24/Principal_Component_Analysis/

[18] G Neelima and Sireesha Rodda. 2016. Predicting user behavior through sessions using the web log mining. In *Advances in Human Machine Interaction (HMI), 2016 International Conference on*. IEEE, 1–5.

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

[20] Pavel Pudil, Jana Novovičová, and Josef Kittler. 1994. Floating search methods in feature selection. *Pattern recognition letters* 15, 11 (1994), 1119–1125.

[21] Stefano Rovetta, Alberto Cabri, Francesco Masulli, and Grażyna Suchacka. 2017. Bot or Not? A Case Study on Bot Recognition from Web Session Logs. In *Italian Workshop on Neural Nets*. Springer, 197–206.

[22] H Nathan Rude and Derek Doran. 2015. Request type prediction for web robot and internet of things traffic. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*. IEEE, 995–1000.

[23] Merve Baş Seyyar, Ferhat Özgür Çatak, and Ensar Gül. 2017. Detection of Attack-Targeted Scans from The Apache HTTP Server Access Logs. *Applied Computing and Informatics* (2017).

[24] Dilip Singh Sisodia and Shrish Verma. 2012. Web usage pattern analysis through web logs: A review. In *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*. IEEE, 49–53.

[25] 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.

[26] Dusan Stevanovic, Aijun An, and Natalija Vlajic. 2012. Feature evaluation for web crawler detection with data mining techniques. *Expert Systems with Applications* 39, 10 (2012), 8707–8717.

[27] Dusan Stevanovic, Natalija Vlajic, and Aijun An. 2013. Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Computing* 13, 1 (2013), 698–708.

[28] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. 2003. CAPTCHA: Using hard AI problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 294–311.

[29] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y Zhao. 2013. You Are How You Click: Clickstream Analysis for Sybil Detection.. In *USENIX Security Symposium*, Vol. 9. 1–008.

[30] Yang Yang, Natalija Vlajic, and UT Nguyen. 2015. Next Generation of Impersonator Bots: Mimicking Human Browsing on Previously Unvisited Sites. In *Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on*. IEEE, 356–361.

[31] Mahdieh Zabihimayvan, Reza Sadeghi, H Nathan Rude, and Derek Doran. 2017. A Soft Computing Approach for Benign and Malicious Web Robot Detection. *Expert Systems with Applications* (2017).