IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON EMERGING APPROACHES TO CYBER SECURITY

Received June 23, 2020, accepted July 22, 2020, date of publication July 30, 2020, date of current version August 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012969

SpiderTrap–An Innovative Approach to Analyze Activity of Internet Bots on a Website

PIOTR LEWANDOWSKI, MAREK JANISZEWSKI, AND ANNA FELKNER
Research and Academic Computer Network–National Research Institute, 01-045 Warsaw, Poland
Corresponding author: Piotr Lewandowski (piotr.lewandowski@nask.pl)

ABSTRACT The main idea behind creating SpiderTrap was to build a website that can track how Internet bots crawl it. To track bots, honeypot dynamically generates different types of the hyperlinks on the web pages leading from one article to another and logs information passed by web clients in HTTP requests when visiting these links. By analyzing the sequences of visited links and passed HTTP requests it is possible to: detect bots, reveal bots' crawling or scanning algorithms, and other characteristic features of the traffic they generate. In our research we focused on identifying and describing whole bots' operations rather than just classifying single HTTP requests. This novel approach has given us insight into what different types of Internet bots are looking for and how they work. This information can be used to optimize the websites for search engines' bots for a better place on a search's results page or prepare a set of rules for tools that filter traffic to the web pages to minimize the impact of bad and unwanted bots on the websites' availability and security. We present the results of the five months of SpiderTrap's activity when honeypot was accessible by two domains (.pl and.eu), as well as by an IP address. The results show examples of activity of well-known Internet bots, such as Googlebot or Bingbot, unknown crawlers, and scanners trying to exploit vulnerabilities in the most popular web frameworks or looking for active webshells (i.e. access points to control a web server left by other attackers).

INDEX TERMS Cyber threat intelligence, honeypot, HTTP, search engines, situational awareness, web crawlers, web search, web spiders.

I. INTRODUCTION

According to the Cisco's white paper [1], all IP traffic in 2017 was about 1.5 zettabytes (that is a 1.5 trillion gigabytes). 17% of the traffic was related to web pages and raw data (excluding all video-related traffic or file sharing, which took 75% and 7% respectively). Cisco estimated annual Internet traffic in 2018 at around 1.8 zettabytes. According to the Distil Networks' report ''2019 Bad Bot Report'' [2] in 2018, 62.1% of all Internet requests they logged were generated by users, 17.5% by good bots, and 20.4% by bad bots. As a bot, we understand a computer program or a script for automatic visiting web pages, where bad bots intend to steal information from the websites or attack them while good bots intend to classify or catalog websites' content following the rules set by the websites' owners.

Bad bots pose different types of threats to the websites. These threats differ from scraping content, through different kind of Denial of Service (DoS) attacks, to overtaking users' accounts or even financial frauds with stolen credit cards' details. Each website owner should assess the possible risks associated with the operation of unwanted bots and implement appropriate security measures.

There are many ways to prevent bad bots from accessing a web page. The most popular of them are: all kinds of the white and black lists of IPs, user agents or HTTP request's features, CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) or scripts monitoring humanlike activity on the web page (i.e. mouse cursor movement) [3].

Most of the good bots are the ones associated with popular search engines like Google, Bing or Yahoo. These bots crawl websites to rank them to present the most relevant results to the search queries. Other type of good bots are the ones run by companies selling Search Engine Optimization (SEO) reports to the websites' owners. SEO reports help with adjusting website's content and its source code to make the website appears higher in search results, attract more traffic and

The associate editor coordinating the review of this manuscript and approving it for publication was Ana Lucila Sandoval Orozco.

finally generate higher revenues to the website's owner. Many SEO techniques available to websites' owners are already described (e.g. [4]) so they will not be an aim of this paper. We would like to focus on the way how good bots crawl the website, how they parse the source code of web pages, whether they recognize different types of links placed on web pages, and finally whether they abide by the restrictions defined in the Robots Exclusion Protocol (described in section II-C).

SpiderTrap was developed to mimic the website and log HTTP requests' headers. The gathered data can be useful for understanding how good bots crawl the websites as well as to prepare the sets of rules for other tools to block the access of bad bots to the legitimate web pages. This paper describes the SpiderTrap architecture and technologies used to develop the honeypot as well as details of data logged through nearly five months of its operation.

The collected information can be used in two perspectives:

- **Research and monitoring.** The collected data – an Internet bots intelligence – can be used to monitor the current level of activity of web bots, both in terms of quantity (for example frequency of requests) and in terms of quality (the way of functioning and behavior). This perspective is mainly covered in the paper.
- **Prevention capability** Thoroughly aggregated, correlated and analyzed requests are used to generate actionable information (some type of indicators of a specific tools or bots). The generated feeds can be used by the Web Application Firewall (WAF) to prevent potentially malicious requests or those that do not abide by the rules and standards.

II. STATE-OF-THE ART

Here, we would like to briefly introduce available solutions for honeypots and website's security that share some similar features with SpiderTrap.

A. HONEYPOTS

There are many different sorts of honeypots available on the Internet. We would like to present just three of them: SNARE with TANNER, BW-Pot and Shadow Daemon as they share some similar features with SpiderTrap.

1) SNARE WITH TANNER

SNARE is an acronym from Super Next generation Advanced Reactive honEypot. It is developed by a non-profit organization MushMush Foundation [5]. This honeypot has a modular architecture. SNARE is a lightweight web server that listens to requests. It passes them to TANNER which decides how to respond, and passes a response back to SNARE which sends it back to the client. TANNER is able to emulate some kind of vulnerabilities, for example: SQL injection, command execution or local file inclusion, using Docker images. It logs information about requests and results of attacks (if it is able to emulate them) in a database. TANNER also clusters requests into sessions to get some statistics about them and to try to determine if the traffic was generated by the attacker, crawler, tool or the user. This honeypot can also copy an existing website to use it as a disguise.

2) BW-POT

BW-Pot is a honeypot build on Docker images with: popular web applications like Wordpress, PhpMyAdmin or Apache Tomcat, with a php webshell, and WOWHoneypot (another simple honeypot). It uses nginx as an HTTP server routing requests to appropriate application. Docker images are reloaded everyday and updated every week for safety. Requests are logged with nginx (HTTP requests) and tshark (raw TCP traffic as.json and.pcap). Logs from nginx and json files from tshark are uploaded via fluentd application to Google BigQuery [6].

3) SHADOW DAEMON

Shadow Daemon is a Web Application Firewall (WAF) with ability of switching to learning mode, thanks to which it can log all requests sent to the honeypot working behind it. For example a honeypot can be a real web application working outside the production environment (in case it would get compromised). Shadow Daemon is able to automatically classify the requests using blacklist and whitelist of rules. Some of the generic rules are predefined but more specific ones must be prepared manually [7].

B. WEBSITE's SECURITY

There are many ways to protect web application and its users. To name a few: extended validity SSL certificates, certificate pinning, HTTP Strict Transport Security (HSTS), firewall, WAF, load balancing with protection against Distributed Denial of Service (DDoS). Some of these methods can be incorporated for free, where some of them are paid services.

1) COMPLEX PAID SOLUTIONS FOR WEBSITE's SECURITY

Imperva, a cybersecurity company, offers a range of products for securing websites and web applications. They have published on their blog an insight into their methods of HTTP requests classification [8], [9]. Their classification method looks for a set of features in incoming HTTP requests: request headers, request patterns, IP address information (such as ASN and domain), or even fingerprinting capabilities of client like for example JavaScript support.

2) SERVICES AND DATABASES PROVIDING ACTIONABLE INFORMATION ABOUT HTTP REQUESTS

Web servers' administrators can also choose to filter HTTP request on their own, using software installed on a server. Requests can be filtered using a firewall, web server, or with a WAF. In the context of website protection, the firewall can block requests from certain IP addresses, the web server can filter HTTP requests by IP address or by matching strings in HTTP headres, and WAF can do all of this as well as inspect

HTTP requests and apply to them all sort of filtering rules which can be very detailed.

Regardless of the chosen solution, there is a need for a reliable data sources to build filtering rules. We would like to discuss here publicly available databases of IP addresses related to malicious or unwanted traffic and databases with User-Agent strings. Both this information (IP addresses and User-Agent strings) can be helpful in securing the website.

3) THE REPUTATION OF IP ADDRESSES

There are two noteworthy community driven projects gathering data about malicious or unwanted activity related to IP addresses – AbuseIPDB [10] and Project Honey Pot [11]. They both aggregate information about malicious or unwanted Internet traffic and process this data to create actionable summaries about IP addresses. Both projects gather similar data, however, AbuseIPDB accepts reports from logged in users and anonymous ones, where Project Honey Pot requires that honeypot software is installed on users' servers in order to send reports. AbuseIPDB provides an API with different limits depending on the user's status or fee, while with Project Honey Pot the user needs to run their honeypot on his own infrastructure in order to access the API. The information provided by AbuseIPDB and Project Honey Pot can be useful for automatically blocking traffic from IP addresses that pose a risk to the website.

4) THE USER AGENTS LISTS

The Hypertext Transfer Protocol [12] specifies that the User-Agent field in the HTTP request header is not mandatory and, if present, it should describe the browsing software and the operating system on which it operates. The HTTP specification does not provide any standard for creating a User-Agent string. It depends on the developer of the software or the software configuration, how the User-Agent string will look like and what data will be given. The User-Agent string can be easily spoofed, so the application can introduce itself as any other to bypass simple filters.

We can indicate two examples of web services for parsing User-Agent strings: www.whatismybrowser.com and www.51degrees.com. Both have large User-Agent strings databases for comparison, and their APIs return as much details as can be obtained from the UA string. These details can be: the name and version of the web browser, the name and version of the underlying operating system, the type of device (PC, smartphone, tablet etc.), and even the capabilities of a web browser, operating system, and device.

An off-line solution, a ''ua-parser'' from www.browserscope.org is a large set or regular expressions for parsing User-Agent strings [13]. The advantage of this solution is being off-line, so there is no need to buy access to any API. However, it is not as robust as the on-line services can be.

C. WAYS OF CONTROLLING BOT ACTIVITY ON A WEBSITE

The Robots Exclusion Protocol (REP) was introduced in the early 1990s. It is not a standard per se, but is widely acclaimed by companies responsibly crawling the Internet. It uses the ''robots.txt'' file, HTML META tags and the X-Robots-Tag in the HTTP header to control the activity of bots on the web page such as: visiting certain web pages, indexing content of them and following links. These methods are suitable for controlling only good bots. It depends on the bots' developers whether their bots will respect such rules or not.

1) DIRECTIVES IN ROBOTS.TXT FILE

The ''robots.txt'' file determines which bots can access which web pages on a website. It can also define the maximum rate at which a bot can crawl a website. The robots.txt file can be useful for excluding some content from being indexed by bots (e.g. static files like pictures or documents), so they will not affect the website ranking in search engines [14]. However, one must use this solution with precaution, as placing links to key pages for the security or operations of the website (i.e. login pages or API) in the ''robots.txt'' makes them easily discoverable to bad actors (bots or humans).

2) META TAGS AND X-ROBOTS-TAGS

The META tags and the X-Robots-Tag HTTP header are returned in the HTML source of the page or in the HTTP response (respectively). Because bots cannot know the values of these tags before visiting a certain web page, this makes this method suitable for controlling whether bots can index the content of a web page and/or follow links on such a page, but they cannot be used to stop bots from visiting certain pages [15].

3) REL ATTRIBUTE IN <a> HTML TAG

Another method of restricting bots from following certain links is to add ''rel'' attribute with the value ''nofollow'' to the link tag in the HTML source code. This method is useful if one wants to exclude only a couple of links [16].

4) BLOCKING BOTS WITH WEB SERVER

The two most popular web servers: Apache and nginx [17] offer the ability to filter certain requests. The most common filtering rules are based on User-Agent string or IP address [18]. It is a simple and efficient way to block bots which do not abide by the Robots Exclusion Protocol, but can be easily recognized by the User-Agent string or IP address. This method fails in the case of highly sophisticated bad bots that connect from the dynamic range of IP addresses with User-Agent strings of legitimate web browsers.

III. SPIDERTRAP - THE TOOL

SpiderTrap is a website developed with a Python framework Django for serving web pages and PostgreSQL as a database for storing requests data. Only valid HTTP requests are logged by Django's middleware. All other requests (e.g. malformed HTTP requests, port scans, etc.) are filtered on nginx HTTP server. There is no TCP/IP packets capturing software. SpiderTrap records HTTP requests' fields (described in details in a subsection below). The gathered data

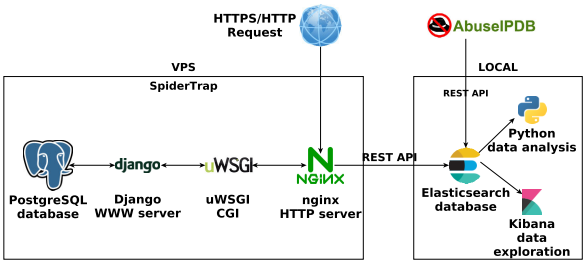

FIGURE 1. Software stack of SpiderTrap.

is analyzed off-line in a system separated from the Virtual Private Server (VPS) on which SpiderTrap is hosted. The analysis takes place in two stages. Firstly, data exploration is performed using Kibana connected to the Elasticsearch instance. In the second step, Python scripts are used to sort data, find interesting features, and plot graphs to visualize them.

A. THE TECHNICAL INFORMATION

SpiderTrap is built from: nginx as HTTP server, uWSGI as the Web Server Gateway Interface for the Django web server and PostgreSQL database for storing HTTP requests coming to SpiderTrap (it is depicted in Fig. 1). The Django framework generates web pages which can be crawled by bots and serves secured API to download the gathered data for analysis. The entire SpiderTrap website is dynamically generated and multilevel. It means that on the main page are links to pages on level 1, on pages of level 1 are only links to pages of level 2 and so on. The content of a website is a set of articles from https://www.gnu.org/philosophy/ with randomly placed links to other articles on SpiderTrap's website. Beside links to the original articles and to the Creative Common license there are no other external links. SpiderTrap can generate an arbitrary number of levels (set by operator), where every page has a set of different types of links leading to the next level of the website. We set maximum level to 3. Below one can find a list of the different types of links. Types 1, 2, and 3 are on every page while 4, 5, and 6 are present only on a main page.

1) standard <a href> links,
2) <a href rel=''nofollow''> links with nofollow attribute,
3) links directing to pages restricted with robots.txt,
4) links in buttons rendered with CSS,
5) links hidden with CSS (they are visible only for plain HTML browsers),
6) links hidden in comments to HTML source of the web page.

This approach makes it possible to analyze how bots crawl the website: horizontally, vertically or in a mixed way. Horizontally means that bot visits all the links found on a certain page, and then starts to visit newly found links visiting all links at a given level. The vertical approach means that the bot visits one link from a certain page, and then visits one link from the newly found and so on until it cannot go deeper. Then it starts from the main page with other found links. The bot can also crawl with a mixed scheme. It collects all found links in the database and then the algorithm decides which link it will visit next.

To track how the bot visits links scraped off the requested pages, information about previously visited pages is encoded in the URL. Each URL consists of a ''slug'', checksum, and type of the link. ''Slug'' is a short title given to an article, it is a term from newspaper publishing. Using slugs in the URLs make them more user-friendly than, for example, using some alphanumeric identifiers. The checksum in the URL is a security measure to prevent crawling a honeypot by fuzzing already crawled links. Fuzzing is a kind of a brute force method where random values are passed to the system to check if it returns some unpredictable results. At the end of each URL a type of the visited link is coded and concatenated with the codes of previously visited links. This makes the URLs looks like this: example.eu/slug-for-an-article,checksum,codedVisitedLinks.

B. CHARACTERISTICS OF REQUESTS GATHERED

SpiderTrap logs the following HTTP request's fields:

- REMOTE_ADDR - client's IP address,
- REMOTE_PORT - TCP connection client port number,
- HTTP_USER_AGENT - string describing the client (e.g. web browser),
- HTTP_REFERER - the URL of the page that referred the requested page,
- HTTP_ACCEPT_ENCODING - list of encoding types accepted by the client,
- HTTP_ACCEPT_LANGUAGE - list of languages used to display content,
- HTTP_ACCEPT - list of formats accepted by the client,
- HTTP_HOST - the name or IP address of the web server specified in the request,
- REQUEST_URI - requested URL,
- REQUEST_METHOD - the method used for the request, e.g. GET, POST, etc.,
- QUERY_STRING - query posted to server with request (if present).

All the most popular web browsers (Chrome, Firefox, Internet Explorer, Opera, Safari) fill these fields in HTTP requests. The more of these fields have no value, the more likely that request is from a bot than a person using a web browser. Although it is possible to create a bot that can send valid values in the HTTP header or even mimic the movement of the mouse pointer and clicks [19], it is not very common. Such sophisticated bots which can outsmart the scripts that protect some web pages from crawlers and scanners are rather targeted for certain websites e.g. e-shops or booking systems.

C. DATA ENRICHMENT

Logs from honeypot can be accessed through the REST API. The API is secured with certificate authentication (nginx) and token (Django). The POST, PUT and DELETE methods were not implemented in the API to minimize the surface of the

TABLE 1. Dataset downloaded from AbuseIPDB API for every IP address logged by SpiderTrap.

Data type	Description
boolean	is IP address a public one
boolean	is IP address whitelisted (i.e. it belongs to a legitimate web crawler, such as Googlebot or Bingbot)
decimal	abuse confidence score (the result of their original algorithm to estimate how malicious is given IP address)
string	country code (ISO alpha-2)
string	Internet Service Provider's (ISP) name
string	domain
string	domain usage type (i.e. hosting, university network, commercial, etc.)

attack. The data is downloaded using a Python script, which enriches it with information from AbuseIPDB (described in Table 1). Data from honeypot API, enriched with data from AbuseIPDB is stored in the local Elasticsearch instance. The illustration of the data analysis configuration is shown in Fig. 1.

IV. THE METHODOLOGY OF DATA ANALYSIS

For the purpose of this paper, we distinguish some kinds of traffic on a standard web page:

1) made by human:
   a) regular web browsing (inoffensive)
   b) reconnaissance (offensive)
   c) exploiting vulnerabilities (offensive)
2) made by bots:
   a) crawling (inoffensive)
   b) scraping (inoffensive, rather unwanted)
   c) reconnaissance (offensive)
   d) exploiting vulnerabilities (offensive)

The main difference between the human and the bot generated traffic is the frequency of requests and the presence of values in the fields: HTTP_USER_AGENT, HTTP_ACCEPT_ENCODING, HTTP_ACCEPT_LANGUAGE, HTTP_ACCEPT and HTTP_HOST. Bots can send requests much more often than people, so rate limiting may be some countermeasure for bots' traffic.

However filtering by presence of values in HTTP's request headers is also effective. Querying the whole dataset for valid requests (i.e. links that web server could resolve) with any value in all fields: HTTP_USER_AGENT, HTTP_ACCEPT_ENCODING, HTTP_ACCEPT_LANGUAGE, HTTP_ACCEPT, and HTTP_HOST shows that only 3960 requests (about 7%) meet this criteria. In this subset, approximately 92% of requests have a signature of MJ12bot in HTTP_USER_AGENT field. Excluding them and some other obvious scanners (e.g. requests where HTTP_HOST is an IP address where it should be a domain) gives a subset of 162 requests (about 0.3% of the whole dataset). This subset can also be narrowed by for example excluding traffic from VPS providers like Amazon AWS but it may also lead to some false positives. Due to fact that our honeypot was not promoted in any way, we assumed it would not attract human visitors or it would be marginal number of requests and it is compliant with this short analysis above.

By analysing the requested URIs, we have prepared a list of rules to classify requests as: inoffensive, offensive or other. Readers should be aware that not all of these rules are general and can not be applied to all of the websites. For example we classify all login panels requests as offensive because there is no such panel on SpiderTrap. However, websites utilizing some well-known web applications like for example: Wordpress, Drupal or PhpMyAdmin will have such panels and not all requests to them will be offensive.

In this paper, we consider requests as inoffensive if they are for the web pages present on a SpiderTrap's website and using the GET, HEAD or OPTIONS methods. Exceptions are requests for files which are not present on SpiderTrap's website but are common among other as they can be useful for bots or applications accessing them. These files are: sitemap.xml, /.well-known/security.txt, /.well-known/assetlinks.json and /.well-known/mta-sts.txt. Another exception are requests for existing web pages with a query string attached by Facebook. This query string consist one parameter "fbclid" with value equal to string of 61 characters (letters, numbers, dashes and underscores) e.g. https://example.com/some_page.html?fbclid=IwAR1mv03LWUaxjWmkCCpYwnrxgrDmfJK8GlCjXosWvh19ZStVAfTI1J_YAnc.

We also do investigate a whole path of links visited by a bot. This information is encoded in the URL (see section III-A). It can be useful to distinguish bots from regular users or check if they behave well. For example requests for links hidden in comment to HTML code of web page or links hidden in invisible element may indicate a bot as these links are not easily accessible to human. It is also possible to check if bot violates robots.txt directives because we can check if particular bot visited restricted link after accessing robots.txt file. List of tags we use for inoffensive requests can be found in Table 2.

As offensive requests we consider these for web pages not present on a SpiderTrap's website and meeting criteria of at least one offensive rule. Rules are based on inspection of three aspects of request: method, query string and URI. Methods: POST, PUT and DELETE are considered as offensive because they are not implemented on SpiderTrap's website and their purpose is to send some data to web server or modify existing one. There is also a PROPFIND method which belongs to Microsoft WebDAV application. This method can be used to scan for vulnerable WebDAV servers so this is why we treat requests with this method as offensive (see Table 3). As SpiderTrap is not processing any input from visitors so most of query strings are also considered as offensive (exception is a query string from Facebook as mentioned above). Another reason to treat query strings as offensive is they can also be an attack vector on the vulnerable web applications. List of strings used to classify query strings can be found in Table 4. To classify requests by the requested URI we propose a list of strings which presence

TABLE 2. A set of rules to classify inoffensive requests by requested URI.

Tag	String to lookup
req_home	/
req_robots	/robots.txt
req_link_restricted	/restricted/
req_sitemap	sitemap.xml
req_security_txt	/.well-known/security.txt
req_assetlinks.json	/.well-known/assetlinks.json
req_mta_scan	/.well-known/mta-sts.txt
req_link_hidden_in_comment	custom regexp
req_link_hidden_in_div	custom regexp
req_link_valid	custom regexp

TABLE 3. A set of rules to classify requests by method.

Tag (classification)	Method
method_post (offensive)	POST
method_put (offensive)	PUT
method_delete (offensive)	DELETE
method_propfind_webdav (offensive) [20]	PROPFIND
method_options (inoffensive)	OPTIONS
method_head (inoffensive)	HEAD
method_other_methods (inoffensive)	other methods

TABLE 4. A set of rules to classify requests by query string.

Tag (classification)	String to lookup
query_facebook_link (inoffensive)	fbclid
query_dl_malware (offensive)	wget, curl
query_path_traversal (offensive)	.. (two dots)
query_thinkphp_scanner (offensive) [21]	think
query_webshell_scan (offensive)	pbid=open, echo, hello
query_other_query (inoffensive)	any other query string

in URI indicates to which group a request belongs. Check up of strings presence in URIs must be done according to the list presented in Table 5 because list's entries are ordered from more specific to the less one. Names of groups (tags) are related to web applications or technologies being a target of attack (beside ''scanner_nmap'' which is related to scanning tool Nmap).

All requests tagged only as req_404 with no other offensive or inoffensive tags are treated as other (for example of such a request see Listing 1).

The aforementioned algorithm for tagging requests is depicted as diagram in Fig. 2 and Fig 3. First diagram presents three algorithms for tagging request's: URI, method, and query independently from each other. Having a list of tags from these three algorithms (for URI, method, and query), there is a last step (depicted in Fig. 3) which classify a request as offensive, inoffensive or other.

For analysis purposes we have defined crawling and scanning sessions. Session is set of at least 2 requests send from 1 IP address with the time between consequent requests being less that 10 seconds. Crawling session must not consist any offensive request while scanning session must consist

```
"http_accept": "*/*",
"http_accept_encoding": null,
"http_accept_language": null,
"http_host": "OUR_DOMAIN",
"http_referer": null,
"http_user_agent": "Mozilla/5.0 (Linux; U;
    Android 2.3.5; ru-ru;
    HTC_IncredibleS_S710e Build/GRJ90)
    AppleWebKit/533.1 (KHTML, like Gecko)
    Version/4.0 Mobile Safari/533.1",
"path_info": "/simpla/",
"query_string": "",
"remote_addr": "62.109.4.125",
"remote_port": 47258,
"request_method": "GET",
"request_uri": "/simpla/",
"timestamp": 1548005142183,
"isIPWhitelisted": false,
"countryName": "Russian Federation",
```

List. 1. Example of the request classified as other.

TABLE 5. A set of rules to classify offensive requests by requested URI.

Tag	String to lookup
scanner_tomcat [22]	/manager/html, jmx-console
scanner_git [23]	.git
scanner_env_file [24]	.env
scanner_nmap [25]	nmaplowercheck
scanner_voip_yealink [26]	y000000000000.cfg, /prov
scanner_voip_asterisk	/servlet
scanner_ncsi [27]	ncsi.txt
scanner_sntp [28]	/html/sntp.html
scanner_horde [29]	/imp/test.php
scanner_weblogic_oracle [30]	bea_wls_deployment_internal
scanner_pma [31]	phpmyadmin, pma, phpma
scanner_wp [32]	wp-, xmlrpc, plugins, wordpress, /wp/
scanner_drupal	drupal
scanner_cgi [33]	cgi-bin, cgi
scanner_mysql	mysql
scanner_sqlite	sqlite
scanner_jboss [34]	.jsp
scanner_sql	sql
scanner_hnap [35]	hnap1
scanner_webdav	webdav
scanner_login	login, admin
scanner_webshell [36]	.php

at least one offensive request. Adjustments of number of requests and time gap is based on the gathered data. We are aware that one IP address can be shared between many clients (networks behind a NAT or many applications working parallel or in a chain) so one address is not always corresponding to one client. Also this approach fails for more sophisticated crawlers or scanners which use multiple instances from IP addresses pools where all of these instances can share the crawled links. There can also be problems with tools using delays longer than 10 seconds between requests (see Fig. 4). With SpiderTrap's ability to serve sites with different sort of links we managed to overcome shortcomings of this method to identify such crawlers and scanners.

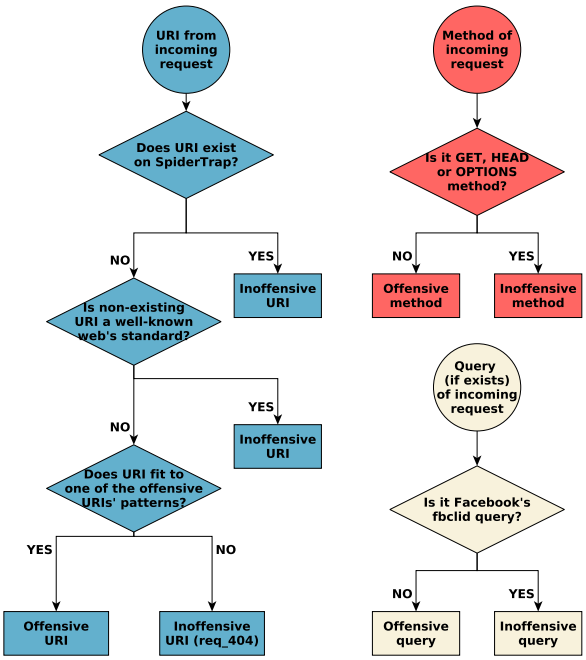

FIGURE 2. Diagram of three algorithms to classify a request's: URI, method and a query as offensive or inoffensive.

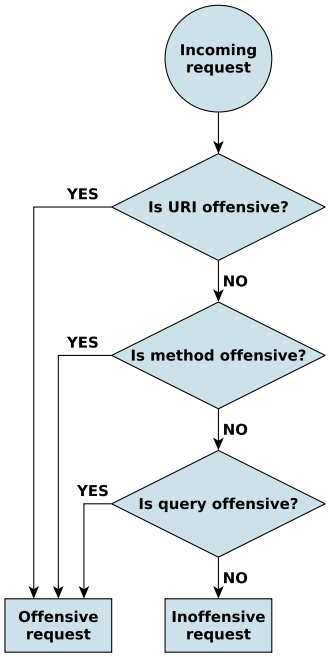

FIGURE 3. Diagram of an algorithm to classify a request as offensive or inoffensive.

V. RESULTS

The raw data of logged requests have been processed with Python scripts. Firstly we tagged requests using a sets of rules to classify them as inoffensive, offensive, or other (as described in the previous section). Then we grouped requests into sessions of scanning or crawling and selected representative examples. These procedures and their results are described in the following subsections.

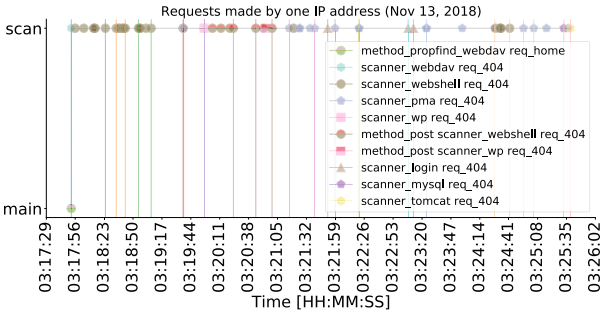

FIGURE 4. Requests sent from 1 IP address on November 13, 2018 seen as 9 different sessions. Every session seats between two vertical lines of the same color. There was one request that did not fall into any of sessions.

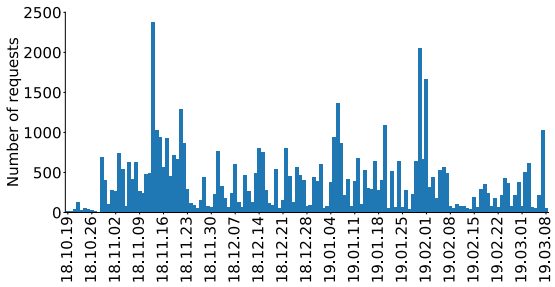

FIGURE 5. Number of requests per day.

A. THE GENERAL RESULTS

SpiderTrap was operating from October 19, 2018, to March 8, 2019. During 140 days, honeypot registered 54054 requests from 6067 distinct IP addresses from 127 countries. Distribution of number of requests per day is depicted in Fig. 5. Most of the requests came from: China, the United States, France and Germany (see Fig. 6). With the list of rules that we have prepared (described in details in section IV), requests were divided into three groups: offensive, inoffensive and others. Groups consist of 33070, 20602 and 382 requests respectively (see Fig. 7). Distribution of requests per group per day is depicted in Fig. 8. Using aforementioned definitions of crawling and scanning sessions we divided the set of requests into sessions. There were 994 crawling sessions and 752 scanning sessions. The number of crawling and scanning sessions per day is depicted in Fig. 9. Fig. 10 and 11 present the number of sessions of scanning and crawling with given number of requests. For both types of sessions the shortest one have 2 requests (as defined above) with the crawling session lasting 0.051 seconds and the scanning session lasting 0.679 seconds. The longest crawling session had 1185 requests (took 510.132 sec) while the longest scanning session had 370 requests (took 385.649 sec).

B. LEGITIMATE, WELL-KNOWN INTERNET BOTS

In Table 6, we present summary for top ten the most active legitimate Internet bots that crawled our honeypot.

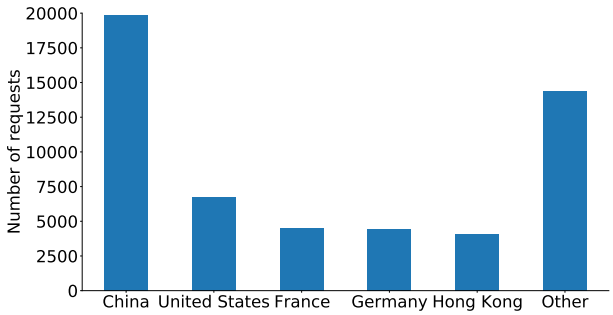

FIGURE 6. Number of requests per country.

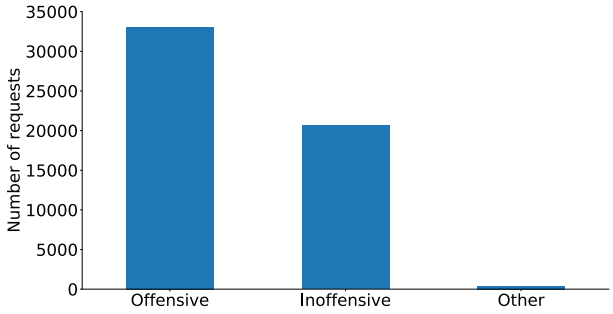

FIGURE 7. Number of requests per group.

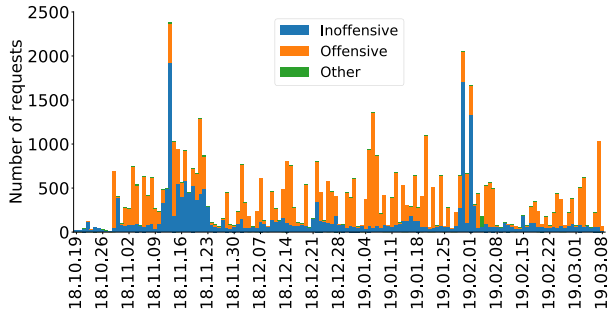

FIGURE 8. Number of requests per group per day.

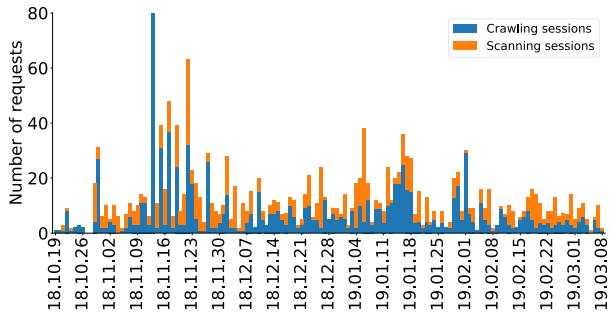

FIGURE 9. Number of crawling and scanning sessions per day.

The number of requests was counted by looking up the User-Agent strings used by these bots (a standard and a mobile versions of the crawlers, if applicable). Number of requests per day is depicted in Fig. 12. The IP addresses from which requests came to the honeypot were checked to see if they can be matched with bots' operators. For 5 bots:

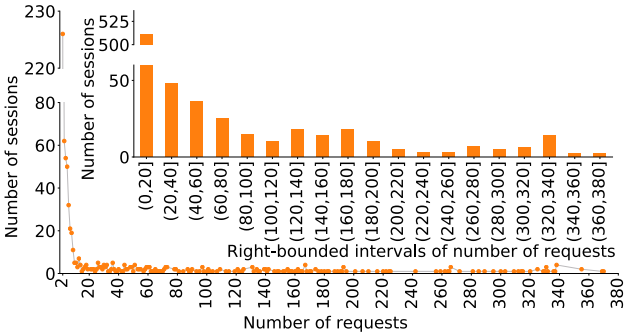

FIGURE 10. Number of scanning sessions with given number of requests. Additional bar chart with histogram of number of scanning sessions with given number of requests.

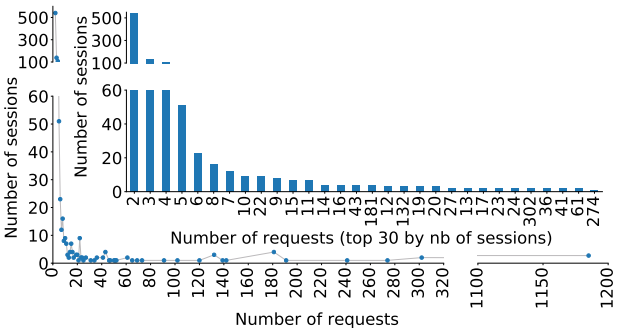

FIGURE 11. Number of crawling sessions with given number of requests. Additional bar chart with top 30 most common number of requests per crawling session.

Googlebot, Bingbot, Ahrefs, YandexBot, and Nimbostratus it was possible, for Dataprovider.com it was partially possible, and for 4 others it was impossible. We also checked if requests with other User-Agent strings came from these IP addresses to exclude obvious User-Agent spoofing. There were only three requests with different User-Agent string (looking like a Chrome browser working on Samsung Galaxy S6 Edge with Android 5.1.1) from IP addresses related to Dataprovider.com.

From these ten bots we choose six to describe them in details as they are best known (like Google or Bing) or presented some unexpected behavior. For other bots, beyond information from Table 6, we observed that: YetiBOT visited only the main page and the first level of sub pages, YandexBot's second user agent was a mobile one (presented itself as an iPhone), and Nimbostratus visited only a main page. Analysis of DotBot's requests did not show any additional features to one presented in Table 6.

1) GOOGLEBOT

Google as the most popular web search engine [17] can enforce new trends in creating websites. Google promotes sites with: enabled communication's encryption (HTTPS), support for mobile devices (through dynamic matching of content to different resolutions and screen sizes), and lastly

TABLE 6. Summary of features of requests made by top ten the most actively crawling SpiderTrap's website legitimate Internet bots.

Atribute	Googlebot	Bingbot	MJ12bot	Ahrefsbot	MegaIndex	Dataprovider	DotBot	YetiBOT	YandexBot	Nimbostratus
Number of requests	2006	999	3896	3767	660	42	664	133	285	104
IP addresses	265	418	51	552	2	3	2	1	3	32
User Agents	3	2	1	1	1	1	1	1	2	1
Respects "robots.txt"	✓	?	✓	✓	X	✓	✓	✓	✓	✓
Does not visit links with the rel="nofollow"	✓	X	X	X	✓	X	X	X	✓	✓
Does not visit links hidden in HTML comments	✓	✓	✓	✓	✓	X	✓	✓	✓	✓
Does not visit links hidden using CSS	X	X	X	X	X	X	X	X	X	✓

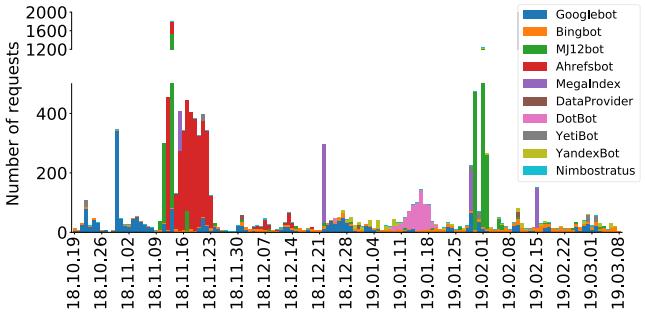

FIGURE 12. Stacked bar graph with number of requests made by crawling bots per day.

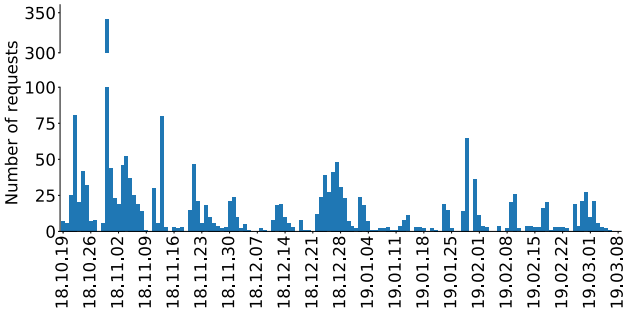

FIGURE 13. Number of requests made by Googlebots per day.

Progressive Web Apps (PWA) – websites which can act as the applications on smartphones. To correctly identify web pages that meet these requirements, Google has three variants of its bot, which present itself as:

- Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
- Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.96 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
- Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.75 Safari/537.36 Google Favicon

The first user agent is a standard Googlebot that crawls pages. It is able to render web pages using CSS and JavaScript. The second is a bot introducing itself as Google Nexus 5X smartphone. It checks if the website's content can dynamically be fitted to the mobile device's screen to be readable. The third, the "Google Favicon" bot crawls the main pages just to download the "favicon", a small picture that can be seen next to the URL in the web browser's address bar or on the quick access page in modern web browsers. Googlebot sometimes requests a page with name consisting of 9 random letters and digits to test if the web server can correctly serve 404 error for a non-existing web page. It also looks for assetlinks.json file. Web developers can put a link to their own or third party's Android applications that support the content of web page [37]. It was also looking for subdomains forum. and homo. (but only for .eu domain). According to our analysis, Googlebot respects the Robots Exclusion Protocol. Fig. 13 presents the number of requests from all types of Googlebots per day.

All recorded IP addresses have been positively verified with the method provided by Google. One must check if reverse DNS lookup of IP address points to domain with "google.com" or "googlebot.com" string. If so, check if the DNS lookup of domain points to the same IP address. For this task we used the host tool available in most of the Linux distributions.

2) BINGBOT

Bing is a web search engine developed by Microsoft. This is the second most popular search engine [17]. Microsoft crawls websites with two different types of bots – standard and a mobile. The mobile Bingbot presents itself as an iPhone. List of User-Agent strings logged by the honeypot:

- Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)
- Mozilla/5.0 (iPhone; CPU iPhone OS 7_0 like Mac OS X) AppleWebKit/537.51.1 (KHTML, like Gecko) Version/7.0 Mobile/11A465 Safari/9537.53 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)

SpiderTrap logged Bingbot's request for a page restricted with robots.txt. The sequence of the surrounding requests is presented in Fig. 14 (link restricted by robots.txt is marked as yellow cirlce). The requests came from three different IP

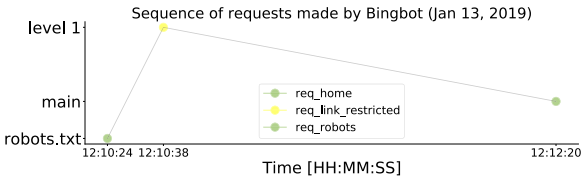

FIGURE 14. Sequence of Bingbot's requests with request for a page restricted with robots.txt.

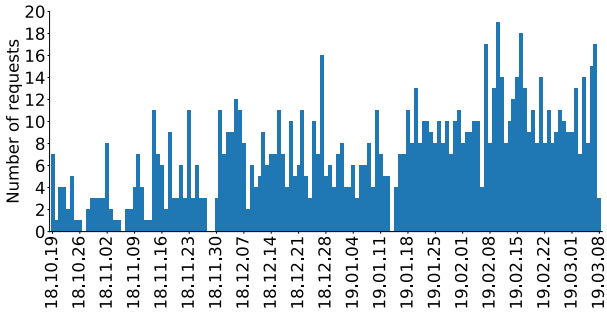

FIGURE 15. Number of requests from Bingbots per day.

addresses. IP addresses have been checked with the Microsoft website [38] and all of them belong to the pool of Microsoft IP addresses for Bingbot. Such a situation took place only once, so it is rather some glitch in bot's operation. It seems that Bingbot does not fully respect the Robots Exclusion Protocol due to visiting links restricted by the "robots.txt" file.

To verify all 418 IP addresses, we tested them using a method similar to that provided by Google. We did reverse DNS check on IP addresses to see if the host tool returns the domains containing "msnbot". We did not use the Microsoft web page as it requires resolving CAPTCHA for each IP address check. A reverse DNS check failed for 2 IP addresses from 13.66.139.0/24 address space. These addresses also failed a check up with aforementioned Microsoft web page. However, the entire address space belongs to Microsoft according to WHOIS. Fig. 15 depicts the number of requests from Bingbot aggregated daily.

3) MJ12bot

MJ12bot is a web crawler developed by Majestic company. According to their web page [39] MJ12bot crawls links to create a map of the Internet with connections between the web pages. Based on this information, Majestic sells Search Engine Optimization (SEO) reports to sites owners. MJ12bot made the largest number of requests among web crawlers. Mostly with five crawls from around 250 to 1450 requests per day, while for the rest of the time it was just periodically visiting a main page and downloading robots.txt file. Visiting links with the rel="nofollow" parameter does not mean that MJ12bot violates the Robots Exclusion Protocol (REP). As long as such links are not taken into account in the ranking of pages, MJ12bot is compliant with REP. The Fig. 16 shows the number of requests per day.

The Majestic web pages [39], [40] inform that MJ12bot is mainly driven by the community. This means that volunteers

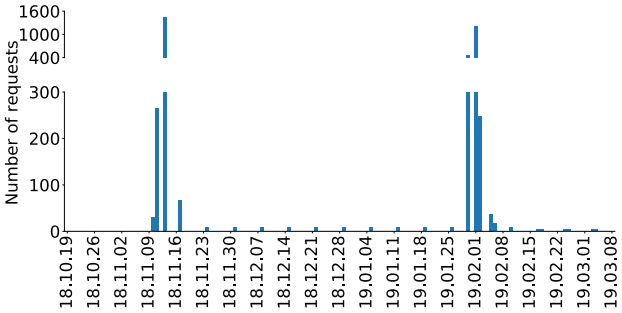

FIGURE 16. Number of MJ12bot's requests per day.

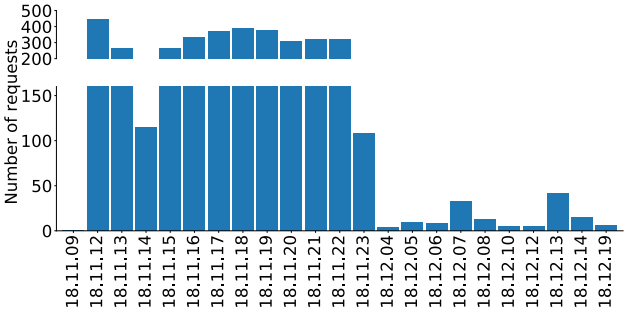

FIGURE 17. Number of AhrefsBot's requests per day.

run MJ12bot instances on their servers to speed up the process of crawling the Internet. However, we could not observe a large diversity of IP addresses. All requests (nearly 3900) were sent only from 51 distinct IP addresses. Comparing this result to Googlebot, one can see that Googlebot generated half of the MJ12bot's traffic using 238 distinct IP addresses. Majestic does not provide any method to verify if a specific IP is a part of their infrastructure or community. The observed User-Agent string was: Mozilla/5.0 (compatible; MJ12bot/v1.4.8; http://mj12bot.com/).

4) AhrefsBot

AhrefsBot is a part of the ahrefs SEO tool. According to Imperva's report from 2017 [41], it was the second most active Internet bot (just behind Googlebot). Our results show that this was the second most active bot on our site just behind MJ12bot (Googlebot was third). The Fig. 17 presents number of requests from AhrefsBot on a given days (there were no requests from this bot on other days). Like with MJ12bot, we can not state if AhrefsBot abides by REP or not.

The Ahrefs's website provides the ranges of IP addresses they are using for crawling [42]. We observed traffic from three subnetworks mentioned on their website (54.36.148.0/24, 54.36.149.0/24, 54.36.150.0/24) and from other subnetworks not mentioned there. We have tested if the addresses which generated traffic are registered in domains pointing "ahrefs.com". All addresses that we have logged passed this test. This bot presents itself with only one User-Agent string: Mozilla/5.0 (compatible; AhrefsBot/5.2; +http://ahrefs.com/robot/).

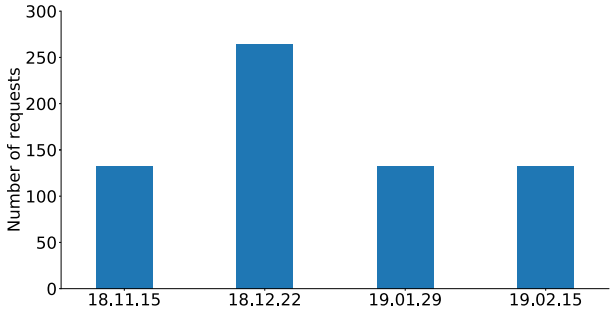

FIGURE 18. Number of requests made by MegaIndex bot on certain dates.

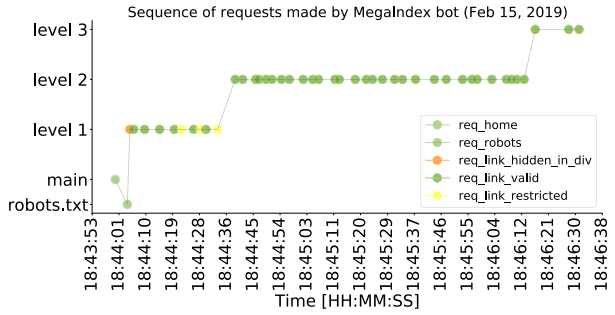

FIGURE 19. Sequence of requests from MegaIndex with requests for links restricted with robots.txt (the yellow one circles).

5) MegaIndex

MegaIndex is another SEO tool with its own web crawler. Like Ahrefs, MegaIndex offers paid SEO reports with analysis of links directing to and from the client's page. Our honeypot has been crawled only four times by MegaIndex (see Fig. 18).

In contrary to the information on the MegaIndex web page [43] we observed violations of robots.txt. However, it visits restricted links only from the main page i.e. this bot does not visit restricted links if it finds them on any other page than the main one. Fig. 19 shows an example of a robots.txt violation. Bot visits the main page and downloads robots.txt and then starts crawling the links found on the main page (marked as "level 1"). The yellow circles represent links restricted with robots.txt. It also visits link hidden in div (presented as an orange circle on a graph).

As stated in the Table 6 we observed only one User-Agent string: Mozilla/5.0 (compatible; MegaIndex.ru/2.0; +http://megaindex.com/crawler) for this bot. Two IP addresses cannot be verified because MegaIndex does not provide any verification method if they really use a given IP addresses. However, we did not observe any other activities from these two IP addresses.

6) DATAPROVIDER

According to the Dataprovider's web page [44], their bot collects all sort of information about websites, such as content (e.g. business information), technical aspects (e.g. payment operator) or even properties of hosting (e.g. details of an SSL certificate). All this data can help business owners get better insight into the visibility of their on-line business as well as compare with businesses of their competitors.

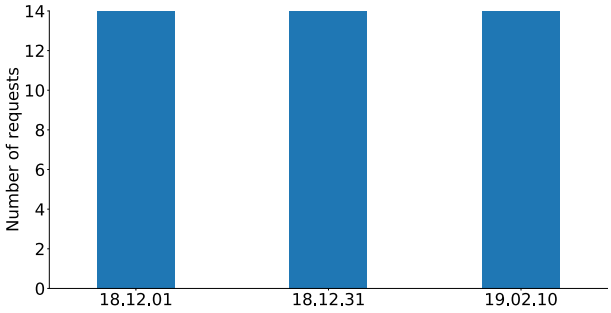

FIGURE 20. Crawls made by Dataprovider.com.

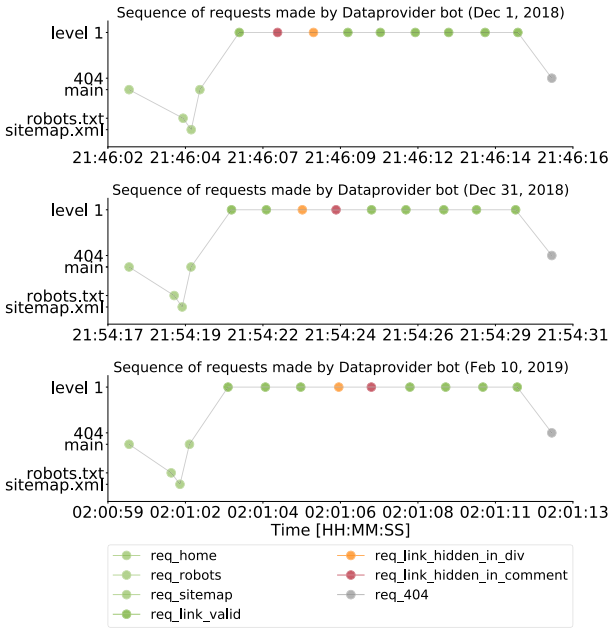

FIGURE 21. Scheme of Dataprovider.com crawl.

The Dataprovider bot was active on SpiderTrap's website only on three days. All of them meet criteria of crawling session. Dates of crawls with number of requests can be seen in the Fig. 20. Interestingly, crawls have identical order of requests. Bot looks for: main page, robots.txt, sitemap.xml, main page (again), nine random links from the main page and the ads.txt file. The ads.txt file is a standard file for web pages that inform advertising networks which companies can advertise on a certain web page. Dataprovider is also one of the few web crawlers which visits links hidden in the comments on the HTML code of the web page. Visiting links hidden in HTML comments does not contradict Robot Exclusion Policy (of course, if such link is not restricted by some rule from "robots.txt"). The Dataprovider's crawling sequences are presented in Fig. 21 (requests for ads.txt file are represented as gray circle labeled req_404).

Dataprovider does not provide any method to verify if the given IP address belongs to them. However, all three IP addresses return a similar domain names. Domains have

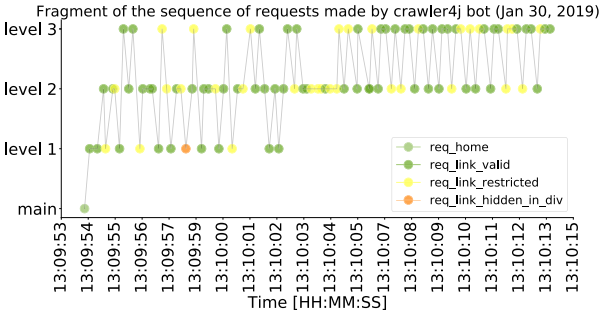

FIGURE 22. First 100 requests from 1184 long crawling session made by crawler4j bot. Gray line is added to make reading a sequence easier.

a distinctive pattern: spider-X.lipperhey.com, where X is a number. The bot's User-Agent string is: Mozilla/5.0 (compatible; Dataprovider.com).

C. CRAWLERS AND SCANNERS

Except requests made by legitimate web crawling bots, we also observed other types of traffic. We can distinguish three main groups:

1) web crawling bots which present themselves as standard browsers
2) web technologies' vulnerabilities scanners
3) other scanners

All three groups with representative examples will be described in the following sections.

1) WEB CRAWLERS

We observed several web crawlers pretending to be a regular web browser and one spoofing legitimate web crawling bot's User-Agent string.

The largest recorded web crawling activity was carried out from one IP address from Canada, using probably the ''crawler4j'' software. This is an open source web crawler written in Java. Two things that make us think it was a crawler4j are, of course, the User-Agent string ''crawler4j (https://github.com/yasserg/crawler4j/)'' and a requests pattern which indicates multi-threading operation. Fig. 22 presents a beginning of this crawling session. After visiting a main page, bot visits couple of links from first and second level and then it visits links from all three levels in random order. We believe this is because one thread generates a pool of links from scrapped web pages while other threads are visiting these links to scrape next ones. This bot crawled 1185 web pages of our honeypot in about 4 minutes.

The second most active crawler was a bot operating from China and pretending to be a multitude of devices (computers and smartphones) working with different versions of web browsers. This bot was operating from October 30, 2018 to February 6, 2019. It performed 964 requests from 270 different IP addresses (all located in China, within 3 subnetworks). This bot was spotted by observing sequences of requests in crawling sessions. We selected crawling sessions starting with requests for links at level 1, 2 or 3 which means the bot had to had these links beforehand, and then we checked if given IP address crawled these links in the past. As the sequence of previously visited kind of links is encoded in URLs (see Table. 7, it is last 6 digits of URL) we could look up for such requests. One of these visited URLs lead us to a possible source. The suspicious request was sent on December 2, 2018 and the only one matching source URL was visited on November 15, 2018. It was sent from a different IP address which also sent requests only for links of level 2 and 3. Going back and forth on the timeline and by checking for correlation in visited URLs and dates of visits, we were able to isolate requests from this bot. For all these requests, the bot used 933 different User-Agent strings. The ten examples of User-Agent strings used by this bot can be found in a Table 11. The first three of them pretend to be a Windows XP (''Windows NT 5.1'') with different web browsers (respectively): Chinese QQ browser and 2 different versions of the Opera. The fourth looks like Safari 5.1.7 web browser running on a 64 bit (''WOW64'') Windows 7 (''Windows NT 6.1''). The rest of the examples are User-Agent strings of different versions of the Chrome web browser running on the Samsung S5 version for the American Sprint Network (''SM-G900P'') with the Android 5.0 ''Lollipop'' operating system. This bot crawled the links restricted with robots.txt (it did not download this file) and the links in the hidden div element, but did not visit the link hidden in the comment to the HTML source code of the main page. It is an example of a sophisticated crawling bot. It used many instances from different IP addresses, it was sharing crawled links between instances and it was pretending to be random users just browsing a website. Nevertheless, thanks to the dynamical generation of the hyperlinks implemented in the SpiderTrap, we were able to detect it.

Another example is a crawler operating from the United States. It used 3 different IP addresses and only one user agent ''Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2)'', which can be interpreted as Internet Explorer 7 working on 64 bits version of Windows XP. This is a very rare set up of operating system and web browser. This bot crawled SpiderTrap's website four times, visiting only the links gathered from the main page (level 1), including the link hidden in the HTML comment (see Fig. 23).

There is also an example of a wrongly implemented web crawler that concatenated the scraped links. The bot was able to successfully scrape links from the main page and the first level, but trying to access the second level of the website, it sent requests to second levels links with the concatenated link of first level at the beginning (see Fig. 24, malformed links are depicted as gray circles). All 132 requests were sent from one IP address in Germany with one user agent ''Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.84 Safari/537.36'' pretending to be Chrome 68 running on Windows 10.

All of these bots did not download the ''robots.txt'' file, so they are not compliant with the Robots Exclusion Protocol.

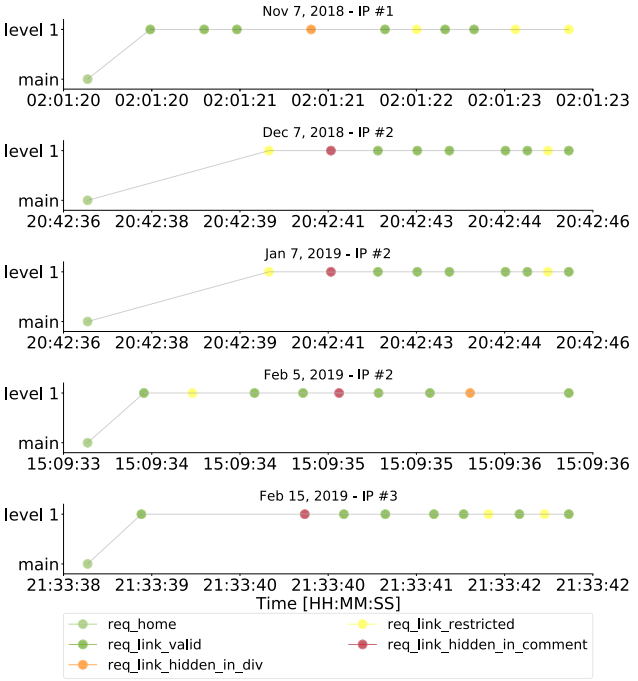

FIGURE 23. Five crawling sessions done by bot presenting itself with very rare User-Agent string: "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2)" and using 3 different IP addresses.

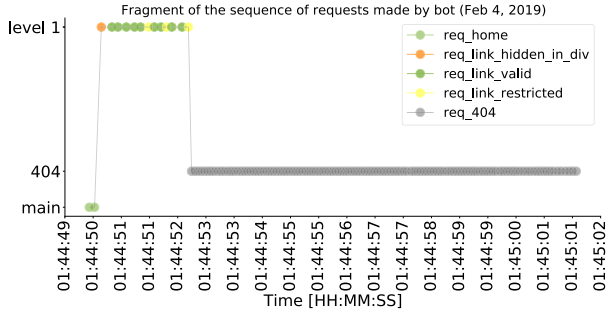

FIGURE 24. Crawling session made by bot with flaw in the source code.

TABLE 7. Example of crawling session (Dec 2, 2018 from 5:48:41 AM to 5:49:06 AM) where bot starts with visiting links of level 3. It visits the same link four times (two times with trailing slash and two without) and then two times the main page.

URL (slug and checksum omitted with '-,-')	User agent
/-,-,325237463057	None
/-,-,325237463057/	None
/-,-,325237463057	Opera/8.0 (Windows NT 5.1; U; en)
/-,-,325237463057/	Opera/8.0 (Windows NT 5.1; U; en)
/	None
/	Opera/8.0 (Windows NT 5.1; U; en)

Even though these bot have only crawled links found on the SpiderTrap web pages, this activity can be seen as unwanted by some site owners.

2) WEB TECHNOLOGIES' VULNERABILITIES SCANNERS

As presented in the Table 5, we were able to pick up from the requested URLs the keywords distinctive for the web facing applications that can be exploited in certain circumstances to get an access to the web server hosting them. Top five the most popular scanning requests were looking for: webshells (14546 requests with POST method and 6870 with GET), PhpMyAdmin login page (6168 GET requests), MySQL login page (1300), other login pages (1233) and Wordpress (799). Statistics for the offensive requests can be found in Table 8.

For example on December 18, 2018, the f5 published an article about attacks on websites that use the ThinkPHP framework [45]. We have observed 91 such scans between December 13, 2018 and January 12, 2019. An example of

TABLE 8. Number of offensive request with given set of tags.

Tags	N
method_post scanner_webshell	14546
scanner_webshell	6870
scanner_pma	6168
scanner_mysql	1300
scanner_login	1233
scanner_wp	799
query_webshell_scan scanner_jboss	258
scanner_git	207
method_propfind_webdav	169
scanner_webdav	165
method_post scanner_wp	144
method_post query_webshell_scan scanner_webshell	142
method_post scanner_sql	132
scanner_weblogic_oracle	110
scanner_sqlite	110
scanner_tomcat	101
scanner_horde	87
query_thinkphp_scanner scanner_webshell	74
scanner_env_file	67
query_other_query scanner_pma	49
method_post scanner_pma	47
method_post	41
scanner_hnap	39
query_other_query scanner_mysql	38
scanner_sql	19
query_dl_malware scanner_webshell	18
query_other_query scanner_login	17
query_other_query scanner_webshell	17
query_other_query	16
scanner_cgi	14
scanner_nmap	10
scanner_voip_yealink	10
query_other_query scanner_voip_asterisk	8
scanner_ncsi	6
query_other_query scanner_wp	6
method_post scanner_login	5
scanner_drupal	4
scanner_jboss	4
method_post scanner_mysql	4
query_dl_malware scanner_cgi	4
scanner_sntp	4
query_other_query scanner_cgi	3
query_other_query scanner_sql	2
query_path_traversal scanner_cgi	2
query_path_traversal scanner_webshell	1
All offensive requests	33070
All offensive sessions	752

TABLE 9. Top 20 the most frequent scanning patterns. ID is corresponding to ID in Figure 26. List of tags is an aggregation of all the tags appearing in given scanning session. Order of appearing of tags as well as number of certain tags differs between sessions with given ID. N is the number of scanning session with given ID.

ID	List of tags (alphabetically sorted and comma separated)	N
1	method_post scanner_webshell req_404	73
2	scanner_wp req_404	57
3	scanner_weblogic_oracle req_404	52
4	req_home, scanner_wp req_404	35
5	scanner_horde req_404	29
6	scanner_git req_404	27
7	req_404, req_home, scanner_pma req_404	26
8	scanner_login req_404, scanner_mysql req_404, scanner_pma req_404, scanner_webshell req_404	21
9	scanner_webshell req_404	18
10	method_post query_webshell_scan scanner_webshell req_404, method_post scanner_sql req_404, method_post scanner_webshell req_404, method_post scanner_wp req_404, method_propfind_webdav req_home, scanner_login req_404, scanner_mysql req_404, scanner_pma req_404, scanner_tomcat req_404, scanner_webdav req_404, scanner_webshell req_404, scanner_wp req_404	17
11	req_404, scanner_login req_404, scanner_pma req_404	17
12	method_post query_webshell_scan scanner_webshell req_404, method_post scanner_pma req_404, method_post scanner_sql req_404, method_post scanner_webshell req_404, method_post scanner_wp req_404, method_propfind_webdav req_home, scanner_login req_404, scanner_mysql req_404, scanner_pma req_404, scanner_tomcat req_404, scanner_webdav req_404, scanner_webshell req_404, scanner_wp req_404	15
13	method_post query_webshell_scan scanner_webshell req_404, method_post scanner_sql req_404, method_post scanner_webshell req_404, method_post scanner_wp req_404, method_propfind_webdav req_home, scanner_login req_404, scanner_mysql req_404, scanner_pma req_404, scanner_webdav req_404, scanner_webshell req_404, scanner_wp req_404	12
14	method_post query_webshell_scan scanner_webshell req_404, method_post scanner_sql req_404, method_post scanner_webshell req_404, method_post scanner_wp req_404, method_propfind_webdav req_home, scanner_pma req_404, scanner_webdav req_404, scanner_webshell req_404, scanner_wp req_404	12
15	scanner_pma req_404, scanner_webshell req_404	11
16	scanner_pma req_404	11
17	method_post query_webshell_scan scanner_webshell req_404, method_post scanner_pma req_404, method_post scanner_sql req_404, method_post scanner_webshell req_404, method_post scanner_wp req_404, method_propfind_webdav req_home, scanner_login req_404, scanner_mysql req_404, scanner_pma req_404, scanner_webdav req_404, scanner_webshell req_404, scanner_wp req_404	10
18	method_propfind_webdav req_home, scanner_pma req_404, scanner_webdav req_404, scanner_webshell req_404	10
19	scanner_login req_404, scanner_mysql req_404, scanner_pma req_404, scanner_tomcat req_404, scanner_webshell req_404	10
20	scanner_login req_404, scanner_pma req_404	9

such HTTP request sent to our honeypot is presented in Listing 25.

Having all requests tagged, we could inspect a list of tags appearing in scanning sessions. We were able to check


```
"http_accept": null,
"http_accept_encoding": null,
"http_accept_language": null,
"http_host": "OUR_IP",
"http_referer": null,
"http_user_agent": "Mozilla/5.0 (Windows NT
    10.0; Win64; x64; rv:61.0) Gecko/20100101
     Firefox/61.0",
"path_info": "/index.php",
"query_string": "s=/index/\\think\\app/
    invokefunction&function=
    call_user_func_array&vars[0]=md5&vars
    [1][]=HelloThinkPHP",
"remote_addr": "181.57.32.173",
"remote_port": 22372,
"request_method": "GET",
"request_uri": "/index.php?s=/index/\\think
    \\app/invokefunction&function=
    call_user_func_array&vars[0]=md5&vars
    [1][]=HelloThinkPHP",
"countryName": "Colombia",
```


FIGURE 25. Details of the HTTP request sent by ThinkPHP's vulnerability scanner.

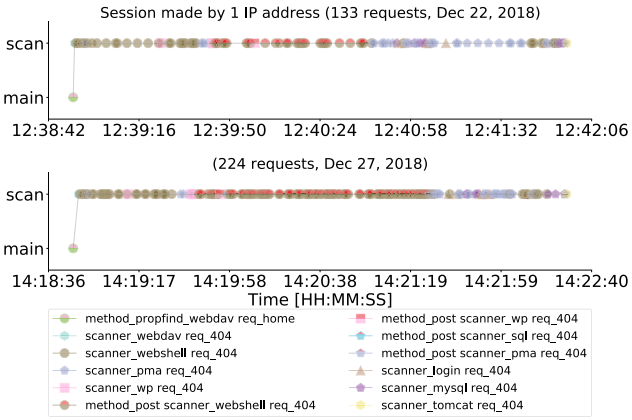

FIGURE 25. Two different sessions made by 1 IP address on December 22 and 27, 2018. There is a difference in a total number of requests and the session made on December 27 has additional requests with tags: method_post scanner_pma req_404 and method_post scanner_sql req_404. It means that scanner was trying to get an access to PhpMyAdmin and other SQL server by sending payload with POST method.

what kind of the scans were chained in scanning sessions and calculate the checksums of the lists of tags to find similar scanning sessions. We observed 752 scanning sessions from 427 unique IP addresses where 79 of them performed more than one scanning session (ranging from 2 to 27 sessions). Of this 79 IP addresses, 32 performed scanning sessions on different days, and 13 of these 32 changed the list of scanned vulnerabilities between scans (for an example of such scans see Fig. 25). The remaining 47 IP addresses performed more than one scanning session on the same dates. Multiple scans from one IP address on the same day can be an effect of two factors: a scanner rerun or the 10 seconds time-out between requests, as described in the definition of the scanning session (see sec. IV). Top 20 the most common

TABLE 10. Details of four scans performed from four different Chinese IP addresses on our honeypot.

N.	Request	HTTP method	User Agent
1	/	PROPFIND	No user agent string (all four the same)
2	/webdav/	GET	Mozilla/5.0 (all four the same)
3-46	.php webshells and misconfigured databases or plugins	GET	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0) Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0) Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:52.0) Gecko/20100101 Firefox/52.0 Mozilla/5.0 (Windows NT 6.1; WOW64; rv:57.0) Gecko/20100101 Firefox/57.0
47-267	.php webshells	POST	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0 Mozilla/5.0 (Windows NT 6.1; WOW64; rv:57.0) Gecko/20100101 Firefox/57.0 Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36 Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
268-338	misconfigured databases	GET	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.105 Safari/537.36 Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0) Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0) Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.84 Safari/537.36

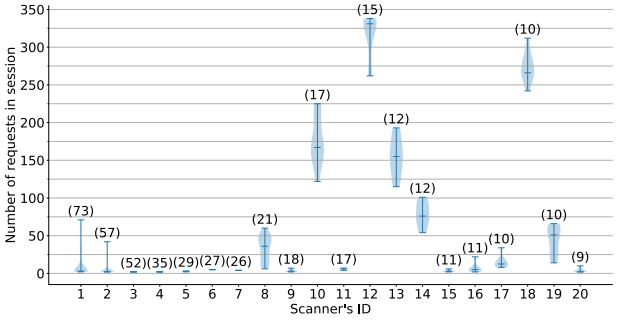

FIGURE 26. Violin plot with top 20 the most common recorded scanning sessions. Every dataset is described by the minimum, maximum and median value of number of requests per scanner's ID. Additionally the contour represents distribution of number of requests. Numbers in parenthesis are the total number of sessions with the given scanner's ID. Descriptions for IDs can be found in Table 9.

types of scanning sessions with information about distribution of number of requests per type are presented in Fig. 26 and in Table 9.

Looking at the scanning sessions with ID 12, we observed that among all 15 scans of this type there were 4 identical scans with 338 requests and 3 with 331, all from 7 different IP addresses (5 from China, 1 from Mexico and 1 from Russia). For example the entire scan with 338 requests can be divided into five parts with different numbers of requests of different types. The first group is one request with the PROPFIND method and without the User-Agent string. The second group is a request for /webdav/ catalog with a rather peculiar User-Agent string "Mozilla/5.0". The third group are 43 requests with the GET method for. php files with webshells, incorrectly configured databases, or plugins for web frameworks like Wordpress or Drupal. The fourth group is the most numerous. It consists of 220 requests using the POST method for. php webshells. The last group has 70 requests for login pages to incorrectly configured databases. Requests from the third, fourth and fifth groups come with the same User-Agent string within the group per scan. During each

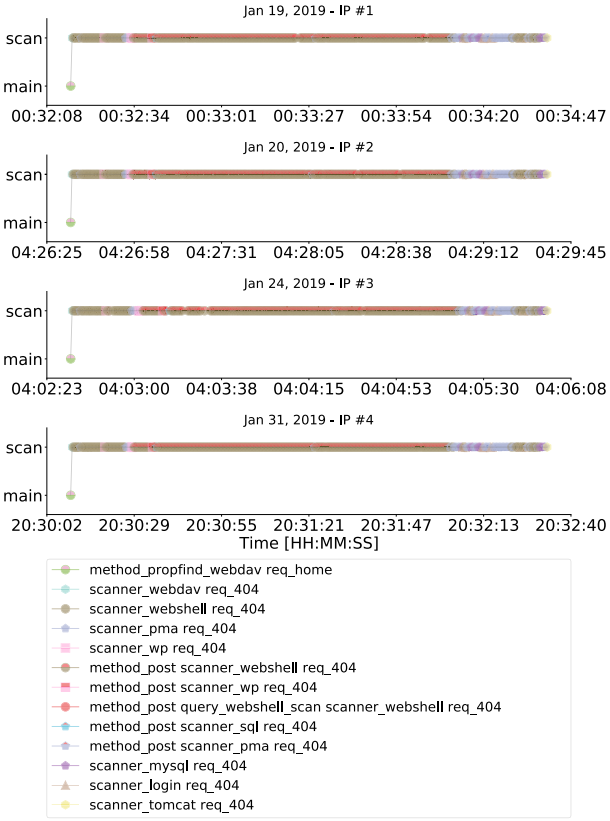

FIGURE 27. Four the same scanning sessions made by 4 different IP address.

scan one can observe five or six different user agents. Graphs presenting these scans can be found in Fig. 27 while details of these scans can be found in the Table 10.

D. USER-AGENT STRING SPOOFING

As stated in the section IV, it is not trivial to distinguish requests sent by bots and people only by checking the HTTP headers, whether the bot uses the legitimate User-Agent string

TABLE 11. Example of ten User-Agent strings used by crawling bot operating from China.

User-Agent strings
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQDownload 732; .NET4.0C; .NET4.0E)
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; en) Opera 9.50
Opera/8.0 (Windows NT 5.1; U; en)
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.57.2 (KHTML, like Gecko) Version/5.1.7 Safari/534.57.2
Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.1548.710 Mobile Safari/537.36
Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.1949.693 Mobile Safari/537.36
Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2211.215 Mobile Safari/537.36
Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.3463.944 Mobile Safari/537.36
Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.3587.376 Mobile Safari/537.36
Mozilla/5.0 (Linux; Android 5.0; SM-G900P Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.3612.707 Mobile Safari/537.36

TABLE 12. Examples of fake user agents.

User agent	Explanation (release dates in parentheses)
Mozilla/5.0 (Windows; U; Windows NT 2.0) Gecko/20091201 Firefox/3.5.6 GTB5	Firefox 3 (2008) on Windows 2000 (2000)
Mozilla/5.0 (Windows NT 6.1; rv:5.0) Gecko/20100101 Firefox/5.0	Firefox 5 (2011) on Windows 7 (2009)
Mozilla/4.0 (compatible; MSIE 6.0;)	Invalid UA of Internet Explorer 6
Mozilla/5.0 (Linux; U; Android 1.5; en-us; sdk Build/CUPCAKE) AppleWebkit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1	Android 1.5 (2009) with reference to Android SDK
Mozilla/5.0 (Linux; U; Android 1.6; es-es; SonyEricssonX10i Build/R1FA016) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1	Android 1.6 (2009)
Mozilla/5.0 (Linux; U; Android 2.2; en-us; Sprint APA9292KT Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1	Android 2.2 (2010)
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.133 Safari/534.16	Chrome 10 (2011) on Windows 7 (2009)
Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.2.28) Gecko/20120306 Firefox/3.6.28	Firefox 3 (2008) on Windows 7 (2009)
Mozilla/1.22 (compatible; MSIE 10.0; Windows 3.1)	Internet Explorer 10 (2012) on Windows 3.1 (1992)
Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)	Internet Explorer 6 (2001) on Windows 98 (1998)
Mozilla/5.0 (Android 4.4; Mobile; rv:41.0) Gecko/41.0 Firefox/41.0	Invalid UA of Firefox 41 (2015) on Android 4.4 (2013)
Mozilla/5.0 (Linux; U; Android 1.5; en-us; T-Mobile G1 Build/CRB43) AppleWebKit/528.5 (KHTML, like Gecko) Version/3.1.2 Mobile Safari 525.20.1	Android 1.5 (2009)
Mozilla/5.0 (Linux; U; Android 2.1-update1; de-de; HTC Desire 1.19.161.5 Build/ERE27) AppleWebKit/530.17 (KHTML, like Gecko) Version/4.0 Mobile Safari/530.17	Android 2.1 (2010)
Mozilla/5.0 (Linux; U; Android 2.2.1; ru-ru; HTC_IncredibleS_S710e Build/FRG83D) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1	Android 2.2.1 (2011)
Mozilla/5.0 (Linux; U; Android 2.2; en-ca; GT-P1000M Build/FROYO) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1	Android 2.2 (2010)
Mozilla/5.0 (Linux; U; Android 2.3.5; ru-ru; HTC_IncredibleS_S710e Build/GRJ90) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1	Android 2.3.5 (2011)
Mozilla/5.0 (Linux; U; Android 4.0.2; en-us; Galaxy Nexus Build/ICL53F) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30	Android 4.0.2 (2011)
Mozilla/5.01694878 Mozilla/5.0 (Windows; U; Windows NT 6.1; en; rv:1.9.2) Gecko/20100115 Firefox/3.6 GTBDFff GTB7.0	Firefox 3 (2008) on Windows 7 (2009)

of a web browser and sends sane values for other fields in the header. Although lists of legitimate user agents are available on the Internet, bot developers tend to automatically generate User-Agent strings using some algorithms. This sometimes lead to a situation in which user agent can be easily recognized as spoofed. For an example let analyze such User-Agent string: ''Mozilla/1.22 (compatible; MSIE 10.0; Windows 3.1)''. ''Mozilla/1.22'' indicates compatibility with old Mozilla version 1.22. ''MSIE 10.0'' implies Internet Explorer version 10, and the ''Windows 3.1'' part is obvious. All this information together, indicates that the user agent is Internet Explorer 10 working on Windows 3.1. It is impossible as Internet Explorer 10 can work only on Windows 7, 8, and 10.

In the Table 12 we present examples of fake User-Agent strings that we observed, together with explanations.

E. MIMICRY OF WELL-KNOWN INTERNET BOTS

We observed only one incident of spoofing the well-known bot's User-Agent string. One bot was impersonating as a Baidu spider. Baidu is a Chinese technology company with services similar to Google. Lack of access to Google services from China (due to the censorship) makes the Baidu search engine the fourth most popular one in the World [17].

SpiderTrap registered 64 requests for the main page from 23 different Chinese IP addresses in two address spaces of /24. These requests came in pairs within about 30 seconds. The first one had a user agent: Mozilla/5.0 (compatible; Baiduspider-render/2.0; +http://www.baidu.com/search/spider.html) and the second had: Mozilla/5.0 (iPhone 84; CPU iPhone OS 10_3_3 like Mac OS X) AppleWebKit/603.3.8 (KHTML, like Gecko) Version/10.0 MQQ

Browser/7.8.0 Mobile/14G60 Safari/8536.25 MttCustomUA/2 QBWebViewType/1 WKType/1. Both requests in pairs came from the same IP address. The second User-Agent string can be interpreted as QQ Browser 7.8 on iOS 10.3. QQ Browser is a web browser written by Tencent, another Chinese technology company (better known of the WeChat application), a Baidu competitor.

According to the Baidu web page [46] the way to check if IP address belongs to Baiduspider's pool is to do a reverse DNS search for this IP address. DNS look up should give a domain with ''.baidu.com'' or ''.baidu.jp''. As we checked, all 23 IP addresses did not have a registered domain and therefore we suppose that this is an example of spoofing well-known bot user agent.

VI. SUMMARY AND FUTURE WORKS

The results of the first version of SpiderTrap are very promising. Logging HTTP requests with a set of basic HTTP headers and knowing relations between links generated on SpiderTrap's website turned out to be enough to track down many sorts of bots, even these more sophisticated. Using a simple list of rules we were able to classify requests as offensive or inoffensive. Merging these two sorts of data gives the ability to create rules applicable to the web servers or web application firewalls to filter out traffic from unwanted bots. At the moment, updated version of SpiderTrap is collecting new data. We would like to check how bots handle the Robots Exclusion Protocol directives sent in HTTP response and META tags in the HTML code. We are also interested in testing the ability of bots to run JavaScript and to store Cookies. The large number of offensive bots detected during this experiment made us to add to SpiderTrap an ability to record the logins and passwords used by these bots to brute force login pages. New version also has implemented the comments section to log the activity of spam bots as well as the content they post on websites. We hope that these enhancements will provide better insight into the operation of wide range of tools targeting websites, not only bots. Moreover, we would like to correlate our results with selected Cyber Threat Intelligence (CTI) feeds to get even more detailed information that could help to improve the automatic methods of fingerprinting and classifying requests coming to the website.

REFERENCES

[1] Cisco, ''Cisco visual networking index: Forecast and trends, 2017–2022,'' Cisco, San Jose, CA, USA, Tech. Rep. 1551296909190103, Feb. 2019. Accessed: May 21, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf

[2] Distil Networks. (2019). *Bad Bot Report 2019: The Bot Arms Race Continues*. Accessed: May 21, 2019. [Online]. Available: https://resources.distilnetworks.com/white-paper-reports/bad-bot-report-2019

[3] Z. Chu, S. Gianvecchio, A. Koehl, H. Wang, and S. Jajodia, ''Blog or block: Detecting blog bots through behavioral biometrics,'' *Comput. Netw.*, vol. 57, no. 3, pp. 634–646, Feb. 2013. Accessed: Apr. 6, 2020. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128612003593

[4] S. Zhang and N. Cabage, ''Search engine optimization: Comparison of link building and social sharing,'' *J. Comput. Inf. Syst.*, vol. 57, no. 2, pp. 148–159, Apr. 2017. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/08874417.2016.1183447

[5] *MushMush*. Accessed: Mar. 26, 2020. [Online]. Available: http://mushmush.org/

[6] K. Isono. (Dec. 2019). *Graneed, BW-Pot*. (in Japanese). Accessed: Mar. 31, 2020. [Online]. Available: https://github.com/graneed/bwpot

[7] *Shadow Daemon Open-Source Web Application Firewall*. Accessed: Mar. 31, 2020. [Online]. Available: https://shadowd.zecure.org/overview/introduction/

[8] O. Cassetto. (Dec. 2014). *Banishing Bad Bots With Incapsula*. Accessed: May 21, 2019. [Online]. Available: https://www.imperva.com/blog/banishing-bad-bots/

[9] E. Roberts. (Sep. 2018). *The Evolution of Hi-Def Fingerprinting in Bot Mitigation*. Accessed: Mar. 31, 2020. [Online]. Available: https://www.imperva.com/blog/the-evolution-of-hi-def-fingerprinting-in-bot-mitigation/

[10] AbuseIPDB. (May 2019). *AbuseIPDB—IP Address Abuse Reports—Making the Internet Safer, One IP at a Time*. Accessed: May 21, 2019. [Online]. Available: https://www.abuseipdb.com/

[11] Project Honey Pot. (May 2019). *The Web's Largest Community Tracking Online Fraud & Abuse | Project Honey Pot*. Accessed: May 21, 2019. [Online]. Available: https://www.projecthoneypot.org/

[12] R. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, document RFC7231, Jun. 2014. Accessed: May 21, 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc7231

[13] (May 2019). *UA-Parser*. Accessed: May 21, 2019. [Online]. Available: https://github.com/ua-parser

[14] Google. (Jan. 2020). *Robots.txt Specifications | Search for Developers*. Accessed: Apr. 3, 2020. [Online]. Available: https://developers.google.com/search/reference/robots_txt

[15] Metatags Company Inc. (May 2019). *How to Use <Meta Name='Robots' Content='Index, Follow'> | the Meaning of Meta Name Robots | Meta Tags Search Engine Promotion*. Accessed: May 21, 2019. [Online]. Available: https://www.metatags.org/meta_name_robots

[16] Google. *Qualify Your Outbound Links to Google—Search Console Help*. Accessed: Apr. 3, 2020. [Online]. Available: https://support.google.com/webmasters/answer/96569?hl=en

[17] StatCounter Global Stats. (May 2019). *Search Engine Market Share Worldwide*. Accessed: May 21, 2019. [Online]. Available: http://gs.statcounter.com/search-engine-market-share

[18] Nocinit. (Jan. 2020). *Bad Bots Blocking—Apache, Nginx & CSF—Tutorial*. Accessed: Apr. 3, 2020. [Online]. Available: https://nocinit.com/blog/bad-bots-blocking-apache-nginx-csf-tutorial/

[19] *Selenium 3.14 Documentation*. Accessed: Mar. 31, 2020. [Online]. Available: https://www.selenium.dev/selenium/docs/api/py/webdriver/selenium.webdriver.common.action_chains.html?highlight=move#selenium.webdriver.common.action_chains.ActionChains.move_by_offset

[20] I. Mathiopoulos. (Mar. 2017). *Number of Internet Facing Vulnerable IIS 6.0 to CVE-2017–7269*. Accessed: Mar. 25, 2020. [Online]. Available: https://medium.com/@iraklis/number-of-internet-facing-vulnerable-iis-6-0-to-cve-2017-7269-8bd153ef5812

[21] A. Remillano. (Jan. 2019). *ThinkPHP Vulnerability Abused by Botnets Hakai and Yowai*. Accessed: Mar. 25, 2020. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/thinkphp-vulnerability-abused-by-botnets-hakai-and-yowai/

[22] *Sensepost/autoDANE*. Accessed: Mar. 25, 2020. [Online]. Available: https://github.com/sensepost/autoDANE

[23] *Git—Getting a Git Repository*. Accessed: Mar. 25, 2020. [Online]. Available: https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository

[24] D. Kundel. (Jan. 2017). *How to Set Environment Variables*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.twilio.com/blog/2017/01/how-to-set-environment-variables.html

[25] G. P. Rodrigues, R. de Oliveira Albuquerque, F. G. de Deus, R. de Sousa, Jr., G. de Oliveira Júnior, L. G. Villalba, and T.-H. Kim, ''Cybersecurity and network forensics: Analysis of malicious traffic towards a honeynet with deep packet inspection,'' *Appl. Sci.*, vol. 7, no. 10, p. 1082, Oct. 2017. Accessed: Mar. 25, 2020. [Online]. Available: http://www.mdpi.com/2076-3417/7/10/1082

[26] Yealink. (Jan. 2019). *Yealink SIP—T2 Series T4 Series T5 Series CP920 IP Phones Administrator Guide*. Accessed: Mar. 25, 2020. [Online]. Available:http://support.yealink.com/previewPdf?file=http

[27] M. Kassner. (Jul. 2011). *What do Microsoft and NCSI Have in Common?* Accessed: Mar. 25, 2020. [Online]. Available: https://www.techrepublic.com/blog/data-center/what-do-microsoft-and-ncsi-have-in-common/

[28] *SNTP—Simple Network Time Protocol (SNTP) Client*. Accessed: Mar. 25, 2020. [Online]. Available: http://doc.ntp.org/current-stable/sntp.html

[29] SonicWall Capture Labs Threat Research Team. (Jan. 2019). *Hackers Actively Scanning for Horde IMP Vulnerability—SonicWall*. Accessed: Mar. 25, 2020. [Online]. Available: https://securitynews.sonicwall.com/xmlpost/hackers-actively-scanning-for-horde-imp-vulnerability/

[30] K. P. Choubey. (Sep. 2019). *Zero Day Initiative—Patch Analysis: Examining a Missing Dot-Dot in Oracle WebLogic*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.thezdi.com/blog/2019/9/16/patch-analysis-examining-a-missing-dot-dot-in-oracle-weblogic

[31] X. Mertens. (Aug. 2017). *Increase of phpMyAdmin Scans*. Accessed: Mar. 25, 2020. [Online]. Available: https://isc.sans.edu/forums/diary/22688/

[32] E. Staff. (Apr. 2016). *Beginner's Guide to WordPress File and Directory Structure*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.wpbeginner.com/beginners-guide/beginners-guide-to-wordpress-file-and-directory-structure/

[33] J. T. Bennett. (Sep. 2014). *Shellshock in the Wild*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.fireeye.com/blog/threat-research/2014/09/shellshock-in-the-wild.html

[34] CISA. (Nov. 2018). *JexBoss–JBoss Verify and EXploitation Tool Analysis Report (AR18-312A)*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.us-cert.gov/ncas/analysis-reports/AR18-312A

[35] R. VandenBrink. (Feb. 2014). *More on HNAP—What is it, How to Use it, How to Find it*. Accessed: Mar. 25, 2020. [Online]. Available: https://isc.sans.edu/forums/diary/17648/

[36] Y. Guo, H. Marco-Gisbert, and P. Keir, "Mitigating webshell attacks through machine learning techniques," *Future Internet*, vol. 12, no. 1, p. 12, Jan. 2020. Accessed: Mar. 25, 2020. [Online]. Available: https://www.mdpi.com/1999-5903/12/1/12

[37] Google. (Mar. 2019). *Asset Links Specification*. Accessed: May 21, 2019. [Online]. Available: https://github.com/google/digitalassetlinks

[38] Microsoft. (May 2019). *Verify Bingbot*. Accessed: May 21, 2019. [Online]. Available: https://www.bing.com/toolbox/verify-bingbot

[39] Majestic. (May 2019). *MJ12Bot | Home | from Majestic*. Accessed: May 21, 2019. [Online]. Available: https://mj12bot.com/

[40] Majestic-12. (May 2019). *Majestic-12: Distributed Search Engine*. Accessed: May 21, 2019. [Online]. Available: https://www.majestic12.co.uk/

[41] I. Zeifman and D. Breslaw. (Feb. 2017). *A Closer Look at the Most Active Good Bots | Imperva*. Accessed: May 21, 2019. [Online]. Available: https://www.imperva.com/blog/most-active-good-bots/

[42] Ahrefs. (May 2019). *What is the List of Your IP Ranges?* Accessed: May 21, 2019. [Online]. Available: https://help.ahrefs.com/getting-started-with-ahrefs/ahrefs-explained/what-is-the-list-of-your-ip-ranges

[43] MegaIndex. (May 2019). *Crawler*. Accessed: May 21, 2019. [Online]. Available: https://megaindex.com/crawler

[44] (May 2019). *Dataprovider.com—We Index the Web and Structure the Data*. Accessed: May 21, 2019. [Online]. Available: https://www.dataprovider.com/

[45] G. Goldshtein. (Dec. 2018). *ThinkPHP 5.x Remote Code Execution Vulnerability*. Accessed: May 21, 2019. [Online]. Available: https://devcentral.f5.com/s/articles/thinkphp-5x-remote-code-execution-vulnerability-32902

[46] Baidu. *FAQs of Baiduspider*. Accessed: Apr. 3, 2020. [Online]. Available: https://help.baidu.com/question?prod_id=99&class=0&id=3001

PIOTR LEWANDOWSKI received the M.Sc. degree in applied physics from the Warsaw University of Technology.

He is currently a Senior Specialist with the Information Security Methods Team, Center of Research and Technology Transfer, Research and Academic Computer Network–National Research Institute. His research interest includes practical aspects of operating systems' security.

MAREK JANISZEWSKI is currently pursuing the Ph.D. degree with the Telecommunication Institute, Warsaw University of Technology.

He is also a Research Associate with the Information Security Methods Team, Center of Research and Technology Transfer, Research and Academic Computer Network–National Research Institute. His research interests include intrusion detection systems, penetration testing, personal data and identity management, and trust and reputation management systems.

ANNA FELKNER received the Ph.D. degree in information technology from the Warsaw University of Technology.

She is currently an Assistant Professor and a Manager with the Information Security Methods Team, Center of Research and Technology Transfer, Research and Academic Computer Network–National Research Institute. Her research interests include cybersecurity systems, risk management, trust frameworks, personal data and identity management, national cybersecurity management, and cooperation in cybersecurity.