Good Bot, Bad Bot: Characterizing Automated Browsing Activity

Xigao Li
Stony Brook University

Babak Amin Azad
Stony Brook University

Amir Rahmati
Stony Brook University

Nick Nikiforakis
Stony Brook University

Abstract—**As the web keeps increasing in size, the number of vulnerable and poorly-managed websites increases commensurately. Attackers rely on armies of malicious bots to discover these vulnerable websites, compromising their servers, and exfiltrating sensitive user data. It is, therefore, crucial for the security of the web to understand the population and behavior of malicious bots.**

In this paper, we report on the design, implementation, and results of Aristaeus, a system for deploying large numbers of "honeysites", i.e., websites that exist for the sole purpose of attracting and recording bot traffic. Through a seven-month-long experiment with 100 dedicated honeysites, Aristaeus recorded 26.4 million requests sent by more than 287K unique IP addresses, with 76,396 of them belonging to clearly malicious bots. By analyzing the type of requests and payloads that these bots send, we discover that the average honeysite received more than 37K requests each month, with more than 50% of these requests attempting to brute-force credentials, fingerprint the deployed web applications, and exploit large numbers of different vulnerabilities. By comparing the declared identity of these bots with their TLS handshakes and HTTP headers, we uncover that more than 86.2% of bots are claiming to be Mozilla Firefox and Google Chrome, yet are built on simple HTTP libraries and command-line tools.

I. Introduction

To cope with the rapid expansion of the web, both legitimate operators, as well as malicious actors, rely on web bots (also known as crawlers and spiders) to quickly and autonomously discover online content. Legitimate services, such as search engines, use bots to crawl websites and power their products. Malicious actors also rely on bots to perform credential stuffing attacks, identify sensitive files that are accidentally made public, and probe web applications for known vulnerabilities [1], [2]. According to recent industry reports [3], bots are responsible for 37.2% of the total website-related traffic, with malicious bots being responsible for 64.7% of the overall bot traffic.

Given the abuse perpetrated by malicious bots, identifying and stopping them is critical for the security of websites and their users. Most existing bot-detection techniques rely on differentiating bots from regular users, through supervised and unsupervised ML techniques, based on features related to how clients interact with websites (*e.g.,* the speed and type of resources requested) [4]–[6] as well as through browser-fingerprinting techniques [7], [8].

In all of the aforementioned approaches, researchers need to obtain a ground truth dataset of known bots and known users to train systems to differentiate between them. This requirement creates a circular dependency where one needs a dataset resulting *from* accurate bot detection to be used *for* accurate-bot detection. The adversarial nature of malicious bots, their ability to claim arbitrary identities (*e.g.,* via User-agent header spoofing), and the automated or human-assisted solving of CAPTCHAs make this a challenging task [9]–[11].

In this paper, we present a technique that sidesteps the issue of differentiating between users and bots through the concept of *honeysites*. Like traditional high-interaction honeypots, our honeysites are fully functional websites hosting full-fledged web applications placed on public IP address space (similar to Canali and Balzarotti's honeypot websites used to study the exploitation and post-exploitation phases of web-application attacks [12]). By registering domains that have never existed before (thereby avoiding traffic due to residual trust [13]) and never advertising these domains to human users, we ensure that any traffic received on these honeysites will belong to benign/malicious bots and potentially their operators. To scale up this idea, we design and build Aristaeus,[1] a system that provides flexible remote deployment and management of honeysites while augmenting the deployed web applications with multiple vectors of client fingerprinting.

Using Aristaeus, we deploy 100 honeysites across the globe, choosing five open-source web applications (WordPress, Joomla, Drupal, PHPMyAdmin, and Webmin), which are both widely popular and have been vulnerable to hundreds of historical vulnerabilities, thereby making them attractive targets for malicious bots. In a period of seven months (January 24 to August 24, 2020) Aristaeus recorded 26.4 million bot requests from 287,017 unique IP addresses, totaling more than 200 GB of raw logs from websites that have *zero organic user traffic.*

By analyzing the received traffic, we discovered that from the 37,753 requests that the average Aristaeus-managed honeysite received per month, 21,523 (57%) were clearly malicious. Among others, we observed 47,667 bots sending unsolicited POST requests towards the login endpoints of our deployed web applications, and uncovered 12,183 unique bot IP addresses which engaged in web-application fingerprinting. In the duration of our experiment, we observed the attempt to exploit five new high-severity vulnerabilities, witnessing bots weaponizing an exploit *on the same day* that it became public.

Furthermore, by analyzing the collected fingerprints and attempting to match them with known browsers and automation tools, we discovered that *at least* 86.2% of bots are lying about their identity, *i.e.,* their stated identity does not match their TLS and HTTP-header fingerprints. Specifically, out

[1] Rustic god in Greek mythology caring over, among others, beekeepers.

of the 30,233 clients, which claimed to be either Chrome or Firefox, we found that 86.2% are lying, with most matching the fingerprints of common Go and Python HTTP libraries as well as scriptable command-line tools (such as wget and curl). Our main contributions are as follows:

- We design and implement Aristaeus, a system for deploying and managing honeysites. Using Aristaeus, we deploy 100 honeysites across the globe, obtaining unique insights into the populations and behavior of benign and malicious bots.
- We extract URLs from exploit databases and web-application fingerprinting tools and correlate them with the requests recorded by Aristaeus, discovering that more than 25,502 bots engage in either fingerprinting, or the exploitation of high-severity, server-side vulnerabilities.
- We curate TLS signatures of common browsers and automation tools, and use them to uncover the true identity of bots visiting our infrastructure. We find that the vast majority of bots are built using common HTTP libraries but claim to be popular browsers. Our results demonstrate the effectiveness of TLS fingerprinting for identifying and differentiating users from malicious bots.

Given the difficulty of differentiating between users and bots on production websites, we will be sharing our curated, bot-only dataset with other researchers to help them improve the quality of current and future bot-detection tools.

II. Background

Unsolicited requests have become a fixture of the web-hosting experience. These requests usually originate from bots with various benign and malicious intentions. On the benign side, search-engine bots crawl websites to index content for their services, while large-scale research projects use bots to collect general statistics. At the same time, malicious actors use bots to identify and exploit vulnerabilities on a large scale. Moreover, high-profile websites are victims of targeted bot attacks that seek to scrape their content and target user accounts.

Bots and automated browsing. An automated browsing environment can be as rudimentary as wget or curl requests, or be as involved as full browsers, controlled programmatically through libraries such as Selenium [14]. The underlying bot platforms and their configuration defines the capabilities of a bot in terms of loading and executing certain resources such as JavaScript code, images, and Cascading Style Sheets (CSS). As we show in this paper, the capabilities and behavior of these platforms can be used to identify them.

Browser fingerprinting. Malicious bots can lie about their identity. Prior work has proposed detection schemes based on browser fingerprinting and behavioral analysis to extract static signatures as well as features that can be used in ML models.

Browser fingerprinting is an integral part of bot detection. Previous research has focused on many aspects of browsing environments that make them unique [15]–[17]. The same techniques have also been used for stateless user tracking by ad networks, focusing on features that are likely to produce different results for different users, such as, the supported JavaScript APIs, list of plugins and browser extensions, available fonts, and canvas renderings [15], [18], [19].

TLS fingerprinting. Similarly, the underlying browser and operating systems can be fingerprinted at the network layer by capitalizing on the TLS differences between browsers and environments [20]. Durumeric et al. used discrepancies between the declared user-agent of a connection and the used TLS handshake to identify HTTPS interception [21]. In this paper, we show how TLS signatures (consisting of TLS version, list of available cipher suites, signature algorithms, e-curves, and compression methods) can be used to identify the true nature of malicious bots, regardless of their claimed identities.

Behavioral analysis. Next to browser and TLS fingerprinting, the behavior of bots on the target website can signal the presence of automated browsing. To that end, features such as the request rate, requests towards critical endpoints (e.g., login endpoint), and even mouse moves and keystrokes have been used by prior bot-detection schemes [4], [5], [22].

Browsing sessions. To study the behavior of bots, we need a mechanism to group together subsequent requests from the same bot. While source IP address can be used for that purpose, in a large-scale study, the IP churn over time can result in false positives where an address changes hands and ends up being used by different entities. To address this issue, we used the notion of "browsing sessions" as used by server-side web applications and also defined by Google Analytics [23]. For each IP address, we start a session upon receiving a request and end it after 30 minutes of inactivity. This allows for an organic split of the active browsing behavior into groups. Grouping requests from the same bot in a session enables us to analyze activities, such as, changes in a bot's claimed identity and multiple requests that are part of a credential, brute-forcing attack.

III. System Design

To collect global bot information, we design Aristaeus, a system that provides flexible honeysite deployment and fingerprint collection. Aristaeus consists of three parts: honeysites, log aggregation, and analysis modules. Our system can launch an arbitrary number of honeysites based on user-provided templates, i.e., sets of existing/custom-made web applications and scripts using virtual machines on public clouds. Aristaeus augments these templates with multiple fingerprinting modules that collect a wide range of information for each visiting client. The information collected from honeysites is periodically pulled by a central server, which is responsible for correlating and aggregating the data collected from all active honeysites. Figure 1 presents the overall architecture of our system.

A. Honeysite Design

A honeysite is a real deployment of a web application, augmented with different fingerprinting techniques, and increased logging. Like traditional honeypots, our honeysites are never advertised to real users, nor linked to by other sites or submitted to search engines for listing. If a honeysite includes publicly-accessible, user-generated content (such as a typical CMS showing blog posts), Aristaeus creates randomly-generated text and populates the main page of the honeysite.

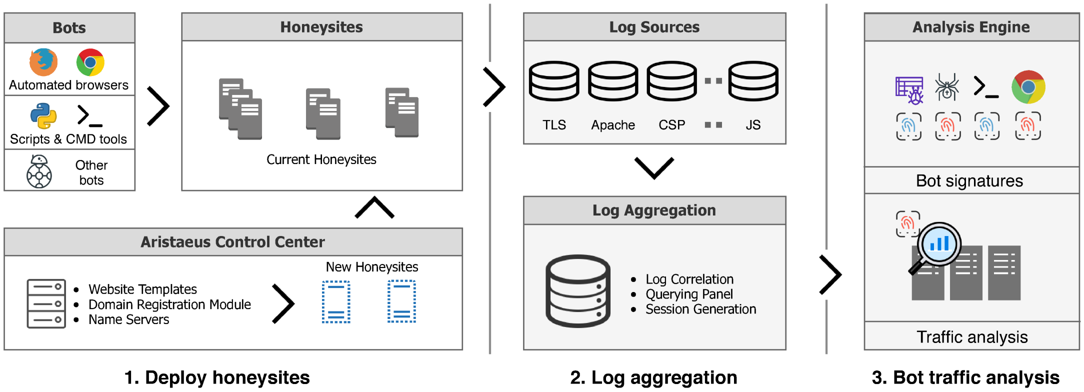

Fig. 1: *High-level overview of the architecture of Aristaeus*

Lastly, to ensure that the traffic a honeysite is receiving is not because its domain name used to exist (and therefore there are residual-trust and residual-traffic issues associated with it [13]) all domains that we register for Aristaeus, were never registered before. To ensure this, we used a commercial passive-DNS provider and searched for the absence of historical resolutions before registering a domain name. As a result, any traffic that we observe on a honeysite can be safely characterized as belonging either to bots, or to bot-operators who decided to investigate what their bots discovered.

To be able to later correlate requests originating from different IP addresses as belonging to the same bot that changed its IP address or from a different deployment of the same automated-browsing software, Aristaeus augments honeysites with three types of fingerprinting: *browser fingerprinting*, *behavior fingerprinting*, and *TLS fingerprinting*. Figure 2 shows how these different types of fingerprinting interface with a honeysite. We provide details and the rationale about each of these fingerprinting modules in the following paragraphs:

1) Browser fingerprinting

To fingerprint each browser (or automated agent) that requests content from our honeysites, we use the following methods.

JavaScript API support. Different bots have different capabilities. To identify a baseline of supported features of their JavaScript engine, we dynamically include an image using *document.write()* and *var img* APIs, and verify whether the connecting agent requests that resource. We also check for AJAX support by sending POST requests from the jQuery library that is included on the page. Lastly, we use a combination of inline JavaScript and external JavaScript files to quantify whether inline and externally-loaded JavaScript files are supported.

Browser fingerprinting through JavaScript. We utilize the Fingerprintjs2 (FPJS2) library [24] to analyze the fingerprintable surface of bots. FPJS2 collects 35 features from web browsers, including information about the screen resolution, time zone, creation of canvas, webGL, and other features that can be queried through JavaScript. To do that, FPJS2 relies on a list of JavaScript APIs. Even though in a typical browsing environment these APIs are readily available, there are no guarantees that all connecting clients have complete JavaScript support. For instance, the WeChat embedded browser is unable to execute *OfflineAudioContext* from FPJS2, breaking the entire fingerprinting script. To address this issue, we modularized FPJS2 in a way that the library would survive as many failures as possible but still collect values for the APIs that are available. Naturally, if a bot does not support JavaScript at all, we will not be able to collect any fingerprints from it using this method.

Support for security policies. Modern browsers support a gamut of security mechanisms, typically controlled via HTTP response headers, to protect users and web applications against attacks. Aristaeus uses these mechanisms as a novel fingerprinting technique, with the expectation that different bots exhibit different levels of security-mechanism support. Specifically, we test the support of a simple Content Security Policy and the enforcement of X-Frame-Options by requesting resources that are explicitly disallowed by these policies [25], [26]. To the best of our knowledge, this is the first time these mechanisms have been used for a purpose other than securing web applications.

2) Behavior fingerprinting

Honoring robots.txt. Benign bots are expected to follow the directives of robots.txt (i.e., do not visit the paths explicitly marked with *Disallow*). Contrastingly, it is well known that malicious bots can not only ignore these Disallow directives but also use robots.txt to identify end-points that they would have otherwise missed [27]. To test this, Aristaeus automatically appends to the robots.txt files of each honeysite Disallow entries to a “password.txt” file, whose exact path encodes the IP address of the agent requesting that file. This enables Aristaeus to not only discover abuses of robots.txt but also identify a common bot operator behind two seemingly unrelated requests. That is, if a bot with IP address 5.6.7.8 requests a robots.txt entry that was generated for a different bot with IP address 1.2.3.4, we can later deduce that both of these requests belonged to the same operator.

Customized error pages. To ensure that Aristaeus gets a chance to fingerprint bots that are targeting specific web applications (and therefore request resources that generate HTTP 404 messages), we use custom 404 pages that incorporate all of the fingerprinting techniques described in this section.

Caching and resource sharing. Regular browsers use caching to decrease the page load time and load resources that are reused across web pages more efficiently. The use

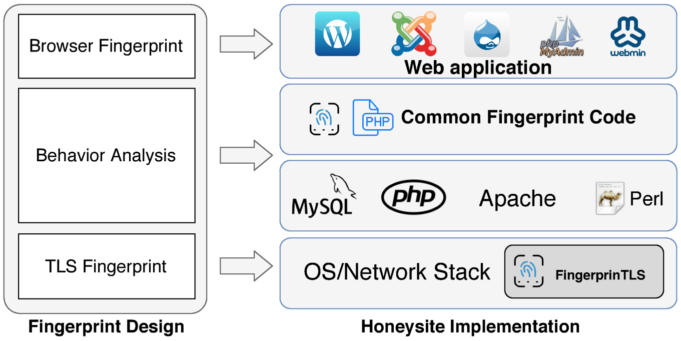

Fig. 2: Internal design of each honeysite.

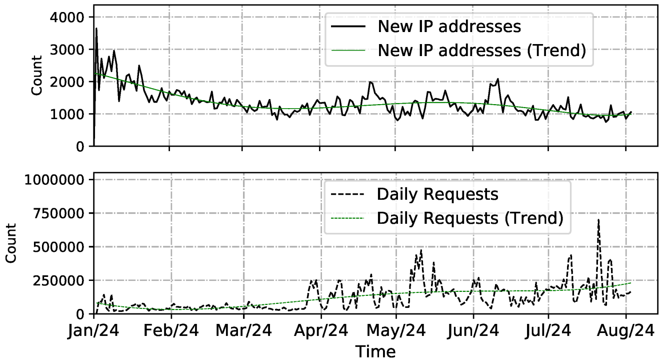

Fig. 3: (Top) Daily number of new IP addresses that visited our honeysites. (Bottom) Daily number of received requests.

of caching can complicate our analysis of logs as we rely on HTTP requests to identify which features are supported by bots. To eliminate these effects, we use the "no-cache" header to ask agents not to cache resources, and we append a dynamic, cache-breaking token at the end of the URLs for each resource on a page.

Cache breakers are generated by encrypting the client IP address and a random nonce using AES and then base64-encoding the output; this makes every URL to the same resource unique. For example, if a bot asks for the resource *a.jpg*, it will send a request in the format */a.jpg?r=[endoded IP+nonce]*. During our analysis, we decrypt the cache breakers and obtain the original IP address that requested these resources. Using this information, we can pinpoint any resource sharing that occurred across multiple IP addresses. This happens when a resource is requested from one IP address but the cache breaker points to a different IP address.

3) *TLS fingerprinting*

We use the fingeprinTLS [20] (FPTLS) library to collect TLS handshake fingerprints from the underlying TLS library of each client that connects to our honeysites over the HTTPS protocol. From each connection, FPTLS gathers fields such as TLS version, cipher suites, E-curves, and compression length. The motivation for collecting TLS fingerprints is that different TLS libraries have different characteristics that can allow us to later attribute requests back to the same software or the same machine. This type of fingerprinting is performed passively (*i.e.,* the connecting agents already send all details that FPTLS logs as part of their TLS handshakes) and can be collected by virtually *all* clients that request content from our honeysites. This is particularly important because, for other types of fingerprinting, such as JavaScript-based fingerprinting, if a bot does not support JavaScript, then Aristaeus cannot collect a fingerprint from that unsupported mechanism.

B. Honeysite Implementation

While Aristaeus is web-application agnostic, for this paper, we deployed five types of popular web applications, consisting of three Content Management Systems (CMSs) and two website-administration tools. On the CMS front, we deployed instances of WordPress, Joomla, and Drupal, which are the three most popular CMSs on the web [28]. WordPress alone is estimated to be powering more than 30% of sites on the web [29]. In terms of website-administration tools, we chose Webmin and PHPMyAdmin. Both of these tools enable administrators to interact with databases and execute commands on the deployed server.

Next to their popularity (which makes them an attractive target), these tools have existed for a number of years and have been susceptible to a large number of critical vulnerabilities, ranging from 49 vulnerabilities (in the case of Webmin) to 333 vulnerabilities (in the case of Drupal). Note that, for CMSs, the vulnerabilities in the core code do not include the thousands of vulnerabilities that third-party plugins have been vulnerable to. Moreover, all five web applications allow users and administrators to log in, making them targets for general-purpose, credential-stuffing bots.

C. Deployment and Log Collection/Aggregation

For the set of experiments described in this paper, we registered a total of 100 domain names. As described in Section III-A, we ensure that the registered domains never existed before (*i.e.,* once registered and left to expire) to avoid residual-traffic pollution. For each domain, Aristaeus spawns a honeysite and deploys one of our web application templates onto the server. We use Let's Encrypt to obtain valid TLS certificates for each domain and AWS virtual machines to host our honeysites over three different continents: North America, Europe, and Asia. Overall, we have 42 honeysites in North America, 39 honeysites in Europe, and 19 in Asia.

A central server is responsible for the periodic collection and aggregation of logs from all 100 honeysites. As described in Section III-A, these include web-server access logs, TLS fingerprints, browser fingerprints, behavior fingerprints, and logs that record violations of security mechanisms. We correlate entries from different logs using timestamps and IP addresses and store the resulting merged entries in an Elasticsearch cluster for later analysis.

IV. Bot Traffic Analysis

In this and the following sections, we report on our findings on the data collected from 100 honeysites deployed across 3 different continents (North America, Europe, and Asia-Pacific) over a 7-month period (January 24, 2020 to August 24, 2020). Overall, our honeysites captured 26,427,667 requests from 287,017 unique IP addresses totaling 207GB of data. A detailed breakdown of our dataset is available in the paper's Appendix (Table VI).

From 26.4 million requests captured, each honeysite received 1,235 requests each day on average, originating from 14 IP addresses. Given that these domains have never been registered before or linked to by any other websites, we consider the incoming traffic to belong solely to bots and potentially the bots' operators.

Daily bot traffic. Figure 3 (Top), shows the number of unique IP addresses that visited our honeysites each day. We observe that this number initially goes down but averages at around 1,000 IP addresses during the later stages of our data collection. Our data demonstrate that our honeysites keep observing traffic from new IP addresses, even 7 months after the beginning of our experiment.

Figure 3 (Bottom) shows the number of requests received over time. Beginning on April 18, we observed an increase in the volume of incoming requests which is not followed by a commensurate increase in the number of unique IP addresses. Upon careful inspection, we attribute this increase in traffic to campaigns of bots performing brute-force attacks on our WordPress honeysites (discussed in more detail in Section VI-A2).

Geographical bot distribution. Although we did not observe significant variation across the honeysites hosted in different regions, we did observe that bot requests are not evenly distributed across countries. Overall, we received most bot requests from the US, followed by China and Brazil. Figure 4 presents the top 10 countries based on the number of IP addresses that contacted our honeysites.

Limited coverage of IP blocklists. We collect information about malicious IP addresses from 9 popular IP blocklists that mainly focus on malicious clients and web threats, including *AlienVault*, *BadIPs*, *Blocklist.de*, and *BotScout*. Out of 76,396 IPs that exhibited malicious behavior against our Aristaeus-managed infrastructure, only 13% appeared in these blocklists.

To better understand the nature of these malicious bots and how they relate to the rest of the bots recorded by Aristaeus, we used the IP2Location lite IP-ASN database [30] to obtain the type of location of all IP addresses in our dataset. Figure 5 shows this distribution. Contrary to our expectation that most bots would be located in data centers, most IP addresses (64.37%) are, in fact, located in residential IP space.

This finding suggests that most bot requests come from either infected residential devices, or using residential devices as a proxy to evade IP-based blocklists [31]. This is also confirmed by the distribution of the IP addresses in the third-party IP blocklists that we obtained. As shown in Figure 5, the vast majority of the hosts labeled as malicious by these blocklists are also located in residential networks.

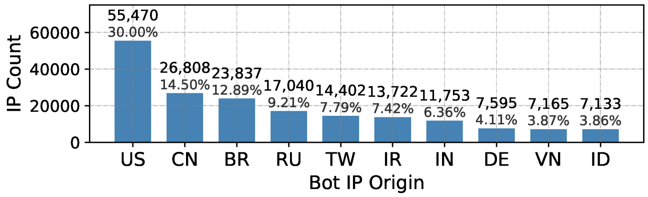

Fig. 4: *Bot Origin Country Distribution.*

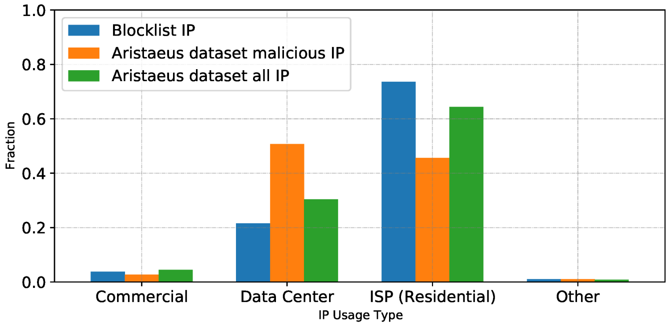

Fig. 5: *Location of IP addresses.*

Lastly, to understand how often the blocklists need to be updated, we characterize the lifetime of bots in our dataset. We define the "lifetime" as the duration between a bot's first visit of a honeysite and the final visit through the same IP address. We observe 69.8% bot IP addresses have a lifetime less than a day. These bots are active for a short period of time and then leave, never to return. In contrast, 0.8% of bots have a lifetime longer than 200 days, approaching the duration of our study. This indicates that bots frequently switch to new IP addresses, which make static IP-based blocklists less effective against IP-churning behavior.

Our comparison of the malicious bots discovered by Aristaeus, with popular blocklists demonstrates both the poor coverage of existing blocklists but also the power of Aristaeus that can identify tens of thousands of malicious IP addresses that are currently evading other tools. We provide more details on our methodology for labeling bots as malicious in Section VI-A.

Honeysite discovery. A bot can discover domain names using DNS zone files, certificate transparency logs, and passive network monitoring. We observe that, mere minutes after a honeysite is placed online, Aristaeus already starts observing requests for resources that are clearly associated with exploitation attempts (e.g. *login.php*, */phpinfo.php*, */db_pma.php*, and */shell.php*). In Section VI-A, we provide more details as to the types of exploitation attempts that we observed in Aristaeus's logs.

To identify how bots find their way onto our honeysites, we inspect the "Host" header in the bot HTTP requests checking whether they visit us through the domain name or by the IP address of each honeysite. Out of the total 287,017 IP addresses belonging to bots, we find that 126,361 (44%) visit us through our honeysite IP addresses whereas 74,728 (26%) visit us through the domain names. The remaining 85,928 (30%) IP addresses do not utilize a "Host" header. Moreover, in HTTPS traffic, we observe that all of 36,266 bot IP addresses visit us through our domain names since Aristaeus does not use IP-based HTTPS certificates.

Given the large volume of requests that reach our honeysites, one may wonder whether we are receiving traffic because of domains that were once configured to resolve to our AWS-assigned IP addresses and whose administrators forgot to change them, when they retired their old virtual machines. This type of "dangling domains" are a source of vulnerabilities and have been recently investigated by Liu et al. [32] and

Borgolte et al. [33]. Using passive DNS, we discovered that two misconfigured third-party domains pointed to our infrastructure, during the duration of our experiment. However, the clients who connected to our honeysites because of these misconfigured domains amount to a mere 0.14% of the total observed IP addresses.

Similarly, to quantify the likelihood that we are receiving requests from real users (as opposed to bots) whose browsers are stumbling upon content linked to via an IP address (instead of through a domain name) back when the Aristaeus-controlled IP addresses used to belong to other parties, we performed the following experiment. We crawled 30 pages for each of the Alexa top 10K websites, searching for content (i.e. images, JavaScript files, links, and CSS) that was linked to via an IP address. Out of the 3,127,334 discovered links, just 31 (0.0009%) were links that used a hardcoded IP address.

When considered together, these findings demonstrate that the vast majority of Aristaeus-recorded visits are not associated with real users, but with bots (benign and malicious) that are the subject of our study.

Most targeted URIs. Not all honeysite endpoints receive the same number of requests. Here, we list the endpoints that received the most requests from bots and the web applications to which they belong.

Among the most requested endpoints, we observe those that are common across multiple web applications, such as *robots.txt*, as well as resources that belong to only one of our web applications, such as *wp-login.php*, which is the login page of WordPress. Figure 6 shows the share of requests for each URI towards different web applications; darker-colored cells indicate a higher portion of requests going towards that type of web application. To enable a baseline comparison, Figure 6 also shows the access distribution of document root (/).

Overall, the login endpoints of our honeysites received the most attention by bots. These requests are brute forcing the login credentials and target endpoints such as *wp-login.php* or */user/login*. There are also general requests for the *robots.txt* file. Interestingly, the *robots.txt* file was mostly queried on WordPress and Joomla honeysites and significantly less for Drupal, PHPMyAdmin, and Webmin honeysites.

Certain resources are only requested on one of our web platforms. For instance, *xmlrpc.php* and *wp-login.php* are only requested on WordPress honeysites. Similarly, the URI */changelog.txt* is requested only from Drupal honeysites for the purpose of web-application fingerprinting (i.e. determining the exact version and whether it is vulnerable to known exploits).

Next is *session_login.cgi* file that hosts the Webmin login page, and we only observe requests to this resource on Webmin instances. Finally, document root and */latest/dynamic/instance-identity/document* requests are observed equally among all of our honeysites. The */latest/dynamic/instance-identity/document* endpoint exists on some of the servers hosted on AWS and can be used in a Server-Side Request Forgery (SSRF) attack (we discuss this attack in Section VI-A2).

Overall, the application-specific attack pattern suggests that bots will fingerprint the applications and identify the presence of a specific vulnerable web application rather than blindly firing their attack payloads. We discuss the fingerprinting attempts by bots in more detail in Section VI-A2. Lastly, Table VII (in the Appendix) lists the most targeted URLs from the perspective of each web application type.

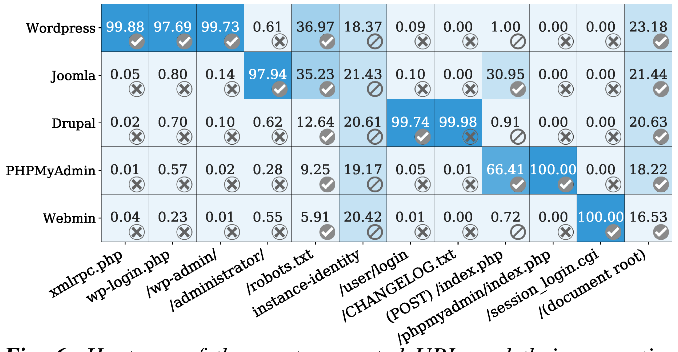

Fig. 6: Heatmap of the most requested URIs and their respective web applications. Darker cells indicate a larger fraction of requests towards that type of web application. The icons in each cell indicate whether the resource is available in that web application. ✓ *indicates that the resource exists;* ⊗ *indicates that the resource does not exist;* ⊘ *indicates that the resource exists but is not available to unauthenticated clients.*

V. JavaScript Fingerprints of Bots

In this section, we report on our findings regarding bot detection through browser fingerprinting.

JavaScript support. We designed several tests to measure the JavaScript support of bots. From these tests, we discovered out that only 11,205 (0.63% of the total 1,760,124 sessions) of bot sessions support JavaScript functionality such as adding dynamic DOM elements and making AJAX requests.

JavaScript-based browser fingerprinting. When it comes to the detection of the bots that visited our honeysites, the effectiveness of JavaScript-based browser fingerprinting is greatly impacted by the lack of support for JavaScript from the majority of bots. Across the total of 1,760,124 sessions, only 0.59% of them returned a JavaScript-based browser fingerprint.

Overall, given that the majority of bots that come to our websites do not support JavaScript, this type of browser fingerprinting proves to be less useful for bot identification. By the same token, this also indicates that if websites demand JavaScript in order to be accessible to users, the vast majority of bots identified by Aristaeus will be filtered out.

VI. Bot Behavior

In this section, we look at different aspects of the behavior of bots during their visits on our honeysites.

Honoring the robots.txt. Based on our dynamic robots.txt generation scheme, we did not observe any violations against Disallow-marked paths. This is an unexpected finding and could be because of the popularity of using fake Disallow entries for identifying reconnaissance attempts [34]–[36]. However, this does not mean that all bots will honor robots.txt, since only 2.2% of total sessions included a request to this file.

Enforcing the content security policy (CSP). Less than 1% of the total IP addresses reported CSP violations on our honeysites. Similarly, less than 1% of bots violated the

CSP rules by loading resources that we explicitly disallowed. The majority of CSP violations originated from benign search-engine bots which were capable of loading embedded resources (such as, third-party images and JavaScript files) but did not support CSP. The vast majority of bots do not load CSP-prohibited resources, not because they honor CSP, but because they do not load these types of resources in general.

Shared/distributed crawling. Since Aristaeus encodes the client's IP addresses into each URL cache-breaker, clients are expected to make requests that match their URLs. However, out of 1,253,590 requests that bore valid cache breakers, we found that 536,258 (42.8%) "re-used" cache-breakers given to clients with different IP addresses.

Given the large percentage of mismatched requests, we can conclude that most are because of distributed crawlers which identify URLs of interest from one set of IP addresses and then distribute the task of crawling across a different pool of "workers". This behavior is widely observed in Googlebots (19.6% of all cache-breaker re-use) and the "MJ12Bot" operated by the UK-based Majestic (32.1% cache-breaker reuse). Interestingly, malicious bots do not engage in this behavior, i.e., any cache-breakers that we receive from them match their IP address.

A. Bot Intentions

Based on their activity on our honeysites, we categorize bot sessions into three categories: "Benign", "Malicious", and "Other/Gray". Benign bots are defined as bots visiting our honeysites and asking for valid resources similar to a normal browser, with no apparent intent to attack our honeysites. For example, benign bots *do not* send unsolicited POST requests nor try to exploit a vulnerability. Contrastingly, malicious bots are those that send unsolicited POST requests towards authentication endpoints, or send invalid requests trying to exploit vulnerabilities. Apart from these two categories, there are certain bots that because of limited interaction with our honeysites, cannot be clearly labeled as benign or malicious. We label these bots as "Other/Gray".

1) Benign bots

Based on their activity, we categorize search-engine bots and academic/industry scanners as benign bots. In total, we recorded 347,386 benign requests, which is 1.3% of the total requests received by Aristaeus.

Search-engine bots. Search-engine bots are responsible for the majority of requests in the benign bots category, and contribute to 84.4% of total benign bots. The general way of identifying search-engine bots is from their User-Agents where they explicitly introduce themselves. However, it is possible for bots to masquerade their User-Agents as search-engine bots in order to hide their malicious activity. Search engines typically provide mechanisms, such as reverse DNS lookups, that allow webmasters to verify the origin of each bot that claims to belong to a given search engine [37]–[40].

In total, we received 317,820 requests from search-engine bots, with Google bots contributing 80.2% of these requests. For instance, we observed four different Google-bot-related user agents ("Googlebot/2.1", "Googlebot-Image/1.0", "AppEngine-Google", and "Google Page Speed Insights") which match documentation from Google [41].

TABLE I: *Breakdown of requests from search engine bots*

Type	Total SEBot Requests	Verified Requests
Googlebot	233,024	210,917 (90.5%)
Bingbot	77,618	77,574 (99.9%)
Baidubot	2,284	61 (0.026%)
Yandexbot	4,894	4,785 (97.8%)
Total	317,820	293,337 (**92.3%**)

Of the 317,820 requests claiming to originate from search-engine bots, we verified that 293,337 (92.3%) are indeed real search-engine bot requests. The share of requests towards our honeysites from the identified search-engine bots are listed in Table I.

Academic and Industry scanners. Apart from anonymous scanners, we identified 30,402 (0.12%) requests originating from scanners belonging to companies which collect website statistics (such as BuiltWith [28] and NetCraft [42]), keep historical copies of websites (such as the Internet Archive [43]), and collect SEO-related information from websites (such as Dataprovider.com). Moreover, the crawlers belonging to a security group from a German university were observed on our honeysites. We were able to verify all of the aforementioned bots via reverse DNS lookups, attributing their source IP address back to their respective companies and institutions.

2) Malicious bots

We define malicious requests by their endpoints and access methods. As we described in Section III-A, we ensure that the domains we register for Aristaeus never existed in the past. Hence, since benign bots have no memory of past versions of our sites, there should be no reason for a benign bot to request a non-existent resource. Therefore, we can label all invalid requests as **reconnaissance** (*i.e.,* fingerprinting and exploitation attempts) requests, which we ultimately classify as malicious. Similarly, we label bots that make unsolicited POST requests to other endpoints, such as login pages, as malicious. Overall, we labeled 15,064,878 requests (57% of total requests) as malicious.

Credential brute force attempts. 13,445,474 (50.8%) requests from 47,667 IP addresses targeted the login page of our websites. By analyzing all unsolicited POST requests we received and checking their corresponding URIs, we discovered that different web applications attract different attacks. For example, there are 12,370,063 POST requests towards WordPress, 90.3% of which are attempts to brute force *wp-login.php* and *xmlrpc.php.* However, for Joomla, there are 343,263 unsolicited POST requests with only 51.6% targeting the Joomla log-in page. The remaining requests are not specific to Joomla and are targeting a wide variety of vulnerable software (e.g. requests towards */cgi-bin/mainfunction.cgi* attack DrayTek devices that are vulnerable to remote code execution [44]).

Interestingly, system management tools attract different patterns of attacks. While 76.2% of POST requests towards

TABLE II: *Top fingerprinting requests*

Path	# requests	Unique IPs	Target applications
/CHANGELOG.txt	116,513	97	Drupal, Joomla, Moodle and spip
/(thinkphp|TP)/ (public|index)	55,144	3,608	ThinkPHP
/wp-content/plugins	32,917	2,416	WordPress
/solr/	23,307	919	Apache Solr
/manager/html	10,615	1,557	Tomcat Manager

phpMyAdmin targeted login endpoints, virtually all POST requests (99.95%) for Webmin targeted its specific login endpoints. This suggests that most bots targeting Webmin focus on brute-forcing credentials, as opposed to targeting other, publicly-accessible pages.

By examining the username and password combinations that were attempted against our honeysites, we observe that attackers always try to login as “admin” using either common passwords [45], or variations of our honeysite domains (i.e. attempting a “www.example.com” password on the honeysite serving the `example.com` domain). From the number of attempts, we found 99.6% of bots (as identified by their IP address) issued fewer than 10 attempts per domain before changing their targets. Only 0.3% of bots issued more than 100 brute-force attempts per domain. The most active bot issued 64,211 login-related requests towards our WordPress honeysites.

Reconnaissance attempts. To identify requests related to reconnaissance, we incorporate a two-prong mapping approach. First, we generate signatures based on popular libraries and databases that include URIs related to *Application fingerprinting*, *Exploitation attempts*, *Scanning for open-access backdoors*, and *Scanning for unprotected backup files*. We provide details for each specific library and dataset later in this section. Second, we manually identify the intention of requests for endpoints that received more than 1,000 requests in our dataset, mapping each request to the aforementioned categories of attacks whenever possible. This filtering step is necessary since we cannot possibly create a comprehensive database that includes signatures for *all* bot requests. As an example of the power of this prioritized-labeling method, via this process we discovered attacks exploiting the recent *CVE-2020-0618* vulnerability in MSSQL Reporting Servers which was not part of our original database of signatures.

Overall, we collected a total of 16,044 signatures, with 179 signatures matching requests in our dataset. These signatures cover 25,502 (9% of total) IP addresses which generated 659,672 requests.

• **Application fingerprinting:** In this study, fingerprinting attempts refer to requests that aim to uncover the presence of specific web-application versions and their plugins. To quantify these requests, we use the signatures from BlindElephant [46] and WhatWeb [47], two open-source fingerprinting tools that have large databases of fingerprints for popular web applications and their plugins. By requesting specific files and matching their content with the signatures in their database, these tools can identify the type and specific version of the target web application. We extract the file paths from the databases of fingerprints and correlate these signatures with our web server logs, to identify fingerprinting attempts from malicious bots. To ensure that we do not label regular crawling as fingerprinting, we discount requests towards generic files, such as, *index.php* and *robots.txt* even if these are valuable in the context of web-application fingerprinting. Overall, our database includes 13,887 URL-based signatures used to identify fingerprinting attempts.

Table II lists the top 5 paths in our database of fingerprinting signatures that received the most requests. In total, we received 223,913 requests that were categorized as fingerprinting attempts and originated from 12,183 unique bot IP addresses. Within our database of signatures, /CHANGELOG.txt has received the largest number of requests since this file can be used to identify the version of Drupal, Joomla, Moodle (Online learning platform), and SPIP (Collaborative publishing system). Second, we observe requests towards remote-code execution (RCE) vulnerabilities in ThinkPHP deployments which are known to be targeted by multiple botnets [48]. The fingerprinting of Apache Solr is related to the versions that were reported to be vulnerable to RCE in November 2019 [49]. Finally, in the top five categories of fingerprinting requests, we observe large numbers of requests towards specific vulnerable WordPress plugins as well as the default deployment of Tomcat Manager. The rest of fingerprinting-related requests follow the same patterns of probing for highly-specific endpoints, belonging to applications that are either misconfigured or known to be vulnerable.

• **Exploitation attempts:** We define exploitation attempts as requests towards URIs that are directly used to trigger known exploits. We use exploits from `exploit-db.com` to generate signatures for exploitation attempts. Unfortunately, automatically generating signatures based on public exploit descriptions is challenging due to the diverse format of vulnerability reports. As a result, we incorporate a human-assisted automation tool that extracts the URLs of signature candidates for the human analyst to verify. At the same time, we hypothesize that bots will most likely focus on *server-side* exploits that are easy to mount (such as SQL injections and RCEs) and therefore focus on these types of exploits, as opposed to including client-side attacks, such as, XSS and CSRF. The resulting signature database includes 593 signatures for the 5 web applications in our dataset corresponding to vulnerabilities from 2014 to 2020. Our database includes 174 exploits for WordPress, 297 exploits for Joomla, 40 for Drupal, 52 for phpMyAdmin, and 19 exploits for Webmin, as well as 14 exploits extracted by looking at the most requested paths on our honeysites.

Overall, we matched 238,837 incoming requests to exploitation attempts that originated from 10,358 bot IP addresses. Table III includes the top 5 endpoints used in these attempts. In this table, we report on the CVE number whenever possible, and in the absence of a CVE number, we report the EDB-ID (Exploit-db ID) for these vulnerabilities.

The RCE vulnerability in PHPUnit received the most exploitation attempts, followed by a setup PHP code injection vulnerability of phpMyAdmin, and an RCE on exposed XDebug servers (PHP Remote debugging tool). Next, an RCE vulner-

TABLE III: *Top exploit requests*

Path	# requests	Unique IPs	CVE/EDB-ID
/vendor/phpunit/ .../eval-stdin.php	70,875	346	CVE-2017-9841
/scripts/setup.php	67,417	1,567	CVE-2009-1151
/?XDEBUG_SESSION _START=phpstorm	23,447	7	EDB-44568
/?a=fetch&content=<php>die(@md5(HelloThinkCMF))</php>	21,819	953	CVE-2019-7580
/cgi-bin/mainfunction.cgi	20,105	2,055	CVE-2020-8515

ability in ThinkCMF (CMS application based on thinkPHP) is also targeted by malicious bots. The last entry in Table III refers to a Draytech vulnerability which is significant in that its exploit was released during our study, allowing us to observe how fast it was weaponized (discussed more in Section VIII).

Interestingly, 3,221 (14%) of IP addresses that sent fingerprinting or exploitation requests were observed in both categories, suggesting that some bots cover a wide range of vulnerabilities, as opposed to focusing on a single exploit.

Next to exploitation attempts, we also searched for requests that included tell-tale shell commands (such as *rm -rf /tmp* and *wget*) in one or more request parameters. In this way, we discovered an additional 24,379 shell-related requests.

Though most injected shell commands attempt to download a malicious payload from a publicly accessible IP address/domain, we discovered that 2,890 requests contain the URL of a private IP address, such as "192.168.1.1:8088" which of course is unreachable from our web servers. These requests could either belong to a buggy bot that extracts the IP address of the wrong network interface after exploiting a host, or could indicate botnets which are meant to attack the routers in a local network, but finally ended up on the public web.

- **Scanning for open-access backdoors:** We generate a list of 485 well-known PHP, ASP, Perl, Java and bash, backdoors. We use the same lists as Starov et al. [50] to extract the signatures of known web backdoors and augment their lists with two repositories that host web shells [51], [52]. Our signatures matched 42 web shells (such as, *shell.php*, *cmd.php* and *up.php*) requested 144,082 times by 6,721 bot IP addresses.
- **Scanning for unprotected sensitive files:** Another group of bots query for unprotected sensitive files, by either guessing the names of likely-sensitive files (such as *backup.sql*) or capitalizing on administrator behavior (e.g. keeping known working copies of sensitive files with a *.old* suffix) and leaks due to specific editors (such as accessing the temporary swap files left behind by Vim).

Similar to web-shell signatures, we used popular word lists used in security scanners, such as SecLists [51], to build a database of 1,016 signatures. These signatures matched 52,840 requests from 5,846 unique bot IP addresses. Files with the *.env* extension which include potentially sensitive environment variables used in Docker as well as other popular development platforms were requested 29,713 times by 1,877 unique bot IP addresses. Bots also requested a wide range of likely sensitive file extensions including *.old*, *.sql*, *.backup*, *.zip*, and *.bak* as well as text editor cache files such as *.php˜* and *.swp* files.

Based on all of our signatures introduced in this section, we observe 929 unique bot IP addresses that participated in all of the aforementioned types of attacks. This demonstrates that there exist bots that are equipped with a large number of exploits and are willing to exhaust them while attacking a host, before proceeding to their next victim.

B. Duration and frequency of bot visits

We grouped the requests recorded by Aristaeus into 1,760,124 sessions. Overall, 44.9% of sessions only consist of a single request. 46% of sessions include between 2-20 requests, whereas there exist 2,310 sessions that include more than 1,000 requests. The majority of bots spend as little as 1-3 seconds on our honeysites. 58.1% of the bots that visited our honeysites left within 3 seconds, and among these bots, 89.5% left within 1 second. Contrastingly, 10.7% of bots spent more than 30 minutes on our honeysites.

A large fraction of bots visiting our honeysites perform too few and too generic requests for us to be able to categorize them as benign or malicious. Specifically, 11,015,403 requests (41.68% of total requests) fall into this category. We provide additional details for these bots below:

Single-shot scanners. 50.04% of the IP addresses that visited our honeysites only sent a single request and did not exhibit any obviously malicious behavior. This is clearly bot behavior since modern browsers make tens of follow-up requests in order to load the required first-party and third-party resources. Similarly, these bots are unlikely to be indexing websites since that would also require follow-up requests for pages linked from the main page. We hypothesize that these single-shot bots are either populating databases for later processing (by more persistent bots) or are searching for specific content that is not present on our setup.

Internet-wide scanners. We attributed 114,489 requests to four different Internet-wide scanners, including Masscan [53] (21.4%) and Zgrab [54] (73.1%). Moreover, our honeysites recorded "Stretchoid" (34.3%) and "NetSystemsResearch" (3.69%) bots, which claim to identify online assets of organizations and the availability of public systems. The exact intention behind these requests remains unclear since these tools can be collecting data for both benign as well as malicious purposes.

C. Unexpected changes in bot identity

In this section, we focus on bots that switched their identity across requests. We look for changes in certain HTTP headers, such as, the user agent, as well as artifacts of a change in the used automation tool, such as, the reordering of HTTP headers.

Multiple User-Agents from the same IP address. At least 14.28% of all IP addresses that visited our honeysites sent requests with two or more user agents. There may be benign reasons why a bot would present two or more User-Agent (UA) strings, such as, bots behind NATs and proxies, or bots that upgraded their versions of browsers/crawling tools. At the same time, we observed clear spoofing behavior, such as, bots changing their UAs with every request. We summarize the types of UA changes below:

1) **Changing the operating system**. As shown in the following example, only the operating system part of the user agent was changed across two requests.
   - ”Mozilla/5.0 (**X11; Ubuntu; Linux x86_64**; rv:52.0) Gecko/ 20100101 Firefox/52.0”
   - ”Mozilla/5.0 (**Windows NT 6.1; WOW64**; rv:52.0) Gecko/ 20100101 Firefox/52.0”

   We identified 5,479 IP addresses that claimed more than one OS during their requests. One possible explanation is that these bots are searching for server-side cloaking, i.e., our honeysites presenting different content to users of Windows vs. Linux.
2) **Changing the browser version.** We use the Levenshtein distance to measure User-Agent string similarity of a certain IP address, recording their minimum and maximum similarity. We observed that when the changes are limited to browser versions as presented in the bots' UAs, the requests exhibit more than 90% similarity. A total of 11,500 IP addresses present these types of version changes. In light, however, of our findings in Section VII-A3 regarding bots imitating common browsers, these version changes are likely to be part of a spoofing strategy, as opposed to honest announcements of browser updates.
3) **Switching user agents frequently** We observed 4,440 (1.54%) IP addresses that sent requests with more than 5 UAs, and there are 542 IP addresses presenting more than 20 UAs across their requests. In extreme cases, we observed 44 IP addresses that sent more than 50 unique UA headers. However, by looking at other HTTP headers and their TLS fingerprint, we can attribute their requests to just one or two tools. This uncovers the strategy of frequently rotating UAs, presumably to evade server-side blocking.

Ordering of HTTP headers. During our analysis of crawling tools, we observed that specific orders of headers can be attributed to usage of certain tools. For instance, we discovered that wget and curl have consistent orderings across versions and are different from each other. By capitalizing on this observation, we identified 23,727 bots that presented more than one header orderings, revealing the usage of more than one tools, regardless of UA claims. Moreover, we discovered 28,627 IP addresses that have only one ordering of HTTP headers, but have multiple UAs, which means they are changing their claimed identities, without changing the underlying tools.

VII. TLS Fingerprinting

Aristaeus serves content to both HTTP and HTTPS requests to accommodate as many bots as possible. Overall, bots made 10,156,305 requests over HTTPS (38.4% of total requests). Out of all HTTPS requests, we extracted 558 unique TLS fingerprints. This indicates that most bots use the same underlying TLS library which is related to the tool and the operating system that they are using. We can therefore use these TLS fingerprints to identify bots and corroborate their claimed identities.

A. TLS Fingerprint of Web Bots

1) Bots behind NAT/Proxy

It is expected that some bots will use proxies before connecting to a website, both in order to evade rate-limiting and blocklisting as well as to potentially distribute the load across multiple servers. Therefore, some requests that originate from the same IP address may be emitted by different bots. To understand whether multiple bots are “hiding” behind the same IP address or whether a single bot is just sending requests with multiple UAs, we can compare the TLS fingerprints across requests of interest. To perform this analysis, we make use of the following observations:

• **Basic Web crawling tools**. Tools like curl and wget will produce only one fingerprint for a given OS and TLS library combination. We use this information to identify the use of these tools in the presence of UA spoofing.

• **Support for GREASE values in the TLS stack**. Chrome, Chromium, and Chromium-based browsers (such as Brave and Opera) support GREASE, a new, TLS-handshake-related, IETF standard [55]. The GREASE values that are sent in the TLS handshakes produce multiple TLS fingerprints across multiple requests, with differences in TLS cipher suites, extensions, and E-curves. As a result, the TLS fingerprint of the aforementioned browsers will be different in the first and the last 1-2 bytes for all GREASE-related handshake values. GREASE was added to Chrome in Version 55 [56] which we verified by testing Chrome on several popular platforms (Ubuntu 16.04, Ubuntu 18.04, CentOS 7, Windows 10, Mac OS, and Android 8).

Contrastingly, browsers such as Firefox, Edge, and Safari have not implemented GREASE at the time of our analysis. As a result, GREASE values are absent from the handshakes and therefore the TLS fingerprints of these browsers remain the same across requests. For these browsers, we collected their fingerprints over different operating systems and used these fingerprints to uncover the true identity of bot requests.

Out of 43,985 IP addresses with a TLS fingerprint, there are 1,396 (3.17%) IPs with two or more sets of TLS fingerprints. For this 3.17%, we observe that the requests originated from different tools and/or OSs, hiding behind the same IP address. If there was a TLS intercepting proxy in place, we would not observe multiple TLS fingerprints but rather a single TLS fingerprint (that of the intercepting proxy). Nevertheless, distinguishing multiple clients behind a TLS proxy remains a challenge for TLS fingerprinting.

2) TLS fingerprint of Tools

Given that only 558 unique TLS fingerprints are shared among all 10,156,305 requests, this means that the majority of requests can be attributed to a small number of tools and TLS libraries.

To perform the matching of bot TLS fingerprints to known tools, we manually extracted the TLS fingerprint of the Go-http-client, wget, curl, Chrome, Firefox, Opera, Edge and IE browsers, and included them in our database of signatures. Moreover, for other TLS fingerprints that we could not reproduce in our setup, we assume that a crawler will not pretend to be another crawler. For example, a crawler built using Python may pretend to be Chrome or Firefox (in order to bypass anti-bot mechanisms), but it has no reason to pretend to be curl given that both tools are well known for building bots and therefore receive the same treatment from anti-bot tools and services [8].

TABLE IV: *Popular TLS fingerprint distribution. Entries below the line correspond to Chromium-based tools that were not in the top ten, in terms of unique bot IP count.*

Tools	**Unique FPs**	**IP Count**	**Total Requests**
Go-http-client	28	15,862	8,708,876
Libwww-perl or wget	17	6,102	120,423
PycURL/curl	26	3,942	80,374
Python-urllib 3	8	2,858	22,885
NetcraftSurveyAgent	2	2,381	14,464
msnbot/bingbot	4	1,995	44,437
Chrome-1(Googlebot)	1	1,836	28,082
Python-requests 2.x	11	1,063	754,711
commix/v2.9-stable	3	1,029	5,738
Java/1.8.0	8	308	1,710
MJ12Bot	2	289	28,065
Chrome-2(Chrome, Opera)	1	490	66,631
Chrome-3(Headless Chrome)	1	80	2,829
Chrome-4(coc_coc_browser)	1	4	101
Total	**113**	38,239	9,879,326

Therefore, after excluding browser TLS fingerprints, we used the description from the majority of recorded UA headers that match the unknown TLS fingerprints, to label them.

Table IV lists the most popular tools covering 113 unique fingerprints. Given that one of these tools is based on Google Chrome, the bottom part of Table IV lists any additional fingerprints that we could trace back to Chrome. The total of these 14 tools produced 9,879,326 requests, covering 97.2% of all TLS requests. Bots using the Go language (and therefore the Go-provided TLS libraries) are by far the most popular, exceeding more traditional choices such as, Python, perl, and wget. We observe a total of four different Chromium-related fingerprints, with distinct fingerprints for bots operated by Google (Googlebot, Google-Image, and Google Page Speed Insights), headless Chrome, and the *coc_coc_browser* corresponding to a Vietnamese version of Chrome.

These results show the power of TLS fingerprinting in corroborating the identity of benign bots and identifying malicious bots that are lying about their identities. Out of 38,312 requests that claimed to be msnbot/bingbot and have a valid TLS fingerprint, we were able to use reverse DNS to verify that all of them were indeed real msnbot or bingbots. Similarily, out of 28,011 requests that claimed to be Googlebot, we matched 27,833 (99.4%) of them through TLS fingerprinting and identify them as real Googlebots. The remaining bots also failed in producing the expected reverse DNS results, pointing to malicious actors who claim the Googlebot identity to avoid getting blocked.

3) Using TLS fingerprinting to uncover the real identity of bots

Given our ability to match claimed user agents (UAs) with presented TLS fingerprints, we checked the TLS fingerprints of all HTTPS-capable bots searching for a mismatch between the stated UAs and the observed TLS fingerprints. Overall, we discovered that 27,860 (86.2%) of the total of 30,233 clients that claim to be Firefox of Chrome, were in fact lying about their identity.

Fake Chrome. Among the 12,148 IP addresses that claimed to be Chrome through their UAs, 10,041 of them do not contain the expected GREASE values in their cipher suites. As a result, we can conclude that more than 82.6% of clients are lying about being Chrome. From their TLS fingerprints, we can conclude that they are mostly curl and wget running on Linux OSs.

Fake Firefox. Similarly, 18,085 IP addresses claimed through their UAs, to be Firefox. However, 12,418 (68.7%) of these Firefox clients actually matched the fingerprints of *Go-http-client*, and 3,821 (21.1%) matched the fingerprints of *libwww-perl*. A small number of requests (5.7%) matched to either python or curl. The remaining 539 IP addresses do not match any of the TLS fingerprints in our database, including the fingerprints of Firefox. Overall, our results show that at least 17,819 out of 18,085 (98.5%) IP addresses that claimed to be Firefox are lying about their identity.

Real Chrome. 351 of the 2,419 IP addresses that show signs of GREASE support in their TLS handshakes, claimed to belong to mobile Safari. This is not possible, given that Safari does not currently support GREASE (neither for Mac nor for the iPhone). This indicates actors (benign or malicious) who wish to obtain the content that websites would serve to mobile Safari browsers, but lack the ability to instrument real Apple devices.

Other TLS fingerprints. Finally, there are 11,693 of bots that have other types of TLS fingerprints, but they mostly belong to Go-http-client, Python-urllib, curl, and wget, as shown in Table IV. They exhibit a wide range of UAs including MSIE, Android Browser, .NET CLR, and MS Word. This indicates a much larger landscape of spoofed client identities, past the Chrome/Firefox spoofing that we investigated in this section.

4) TLS fingerprints in Exploitation attempts

We applied our method of matching TLS fingerprints to the stated identities of the bots behind the malicious requests we previously discussed in Section VI-A2. Table V presents the results. First, we can observe that there are almost no real browsers accessing those resources, corroborating our exploitation labels (under the reasonable assumption that attackers do not need full-fledged browsers to send malicious payloads to vulnerable websites). Second, there are major variations in the different type of malicious requests. For example, 93.4% of exploit requests are using Golang, but only 171 requests are using Golang to look for misplaced Backup files. Similarly, libwww/wget is popular in the backdoor requests, but these tools do not appear in backup file probing requests. These results indicate different generations of tools and attackers, using different underlying technologies to exploit different website vulnerabilities.

VIII. Case Studies

Bots only focusing on JS resources. Even though many bots do not request images and other resources (presumably as a way of speeding up their crawls) we observed bots that *only* request JavaScript resources. One bot in our dataset (IP address: 101.4.60.1**) was particularly interesting, as it only downloaded JavaScript files but never, according to Aristaeus' tests, executed them. Given that the IP address of this bot belongs to a Chinese antivirus company, we suspect that the intention of that bot is to collect JavaScript files for anti-malware research purposes.

TABLE V: TLS fingerprint of malicious requests

Type	Python	Golang	libwww / wget	Chrome / Firefox	Unknown	Total
Backdoor	231	1,718	349	3	482	2,783
Backup File	411	171	84	0	1,803	2,469
Exploits	275	18,283	607	0	390	19,555
Fingerprinting	1,524	3,670	630	139	7,226	13,189

Spikes in incoming traffic. We observe two major spikes in our dataset. The first traffic surge happened from May 28th to June 17th, where a group of bots continuously sent us log-in attempts and XML-RPC requests. These bots initially requested */wp-includes/wlwmanifest.xml*) to check if a honeysite was an instance of WordPress. They then extracted the list of users from the author-list page, and then started brute-forcing the admin account through POST requests towards `xmlrpc.php` (targeting WordPress's authentication point that is meant to be used as an API). This group of bots issued a total of 4,851,989 requests, amounting to 18.4% of the total requests. Similarly, the second traffic surge corresponds to 21.9% of the total requests in our dataset.

Failed cloaking attempts. Modifying the HTTP user agent header is likely the most common method of cloaking used by the bots (both malicious bots trying to exploit websites as well as benign bots operated by researchers and security companies). Yet during our study, we observed failed attempts to modify this header. For instance, we observed wrong spellings of the "User-Agent" header including "useragent" and "userAgent". Similarly, the "Host" header also included different spellings and letter cases, such as "HOST", "host", or "hoSt". The appearance of these spelling artifacts means that these header fields are forged. For certain HTTP libraries however, an incorrect spelling results in both the original header and the new header being sent out. Therefore, some requests recorded by Aristaeus included both "User-Agent" and "userAgent" headers. For these bots, the original "User-Agent" header indicated, e.g., "*python-requests/2.23.0*", whereas the "userAgent" header reported "*Mozilla/5.0 (Windows NT 6.1; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0*".

Time to weaponize public exploits. During the seven-month span of this study, we observed requests that tried to exploit five remote command execution (RCE) vulnerabilities that went public *after* the start of our data collection. As a result, we have visibility over the initial probes for these exploits. The five RCE vulnerabilities affect the following software/firmware: DrayTech modems (CVE-2020-8585), Netgear GPON router (EDB-48225), MSSQL Reporting Servers (CVE-2020-0618), Liferay Portal (CVE-2020-7961), and F5 Traffic Management UI (CVE-2020-5902).

For the first vulnerability on DrayTech devices, the exploit was released on March 29, 2020 and we observed exploitation attempts on Aristaeus' honeysites a mere two days later.

In a similar fashion, the exploit for Netgear devices went public on March 18 2020, and the first exploitation attempts were recorded by Aristaeus *on the same day*. Next, the proof-of-concept exploit for MSSQL reporting server vulnerability went public by the researcher who reported this vulnerability on February 14, 2020, and we received exploitation attempts for this vulnerability 4 days later [57]. The Liferay vulnerability went public on March 20, 2020 and exploiting requests showed up in Aristaeus' logs after 4 days. Finally, the F5 vulnerability was publicly announced on June 30, 2020 and we observed requests towards F5 TMUI shell *on the same day*.

Based on these five occasions, we can clearly observe that the time window between an exploit going public and malicious actors probing for that vulnerability is short and, in certain cases (such as the Netgear and F5 devices) non-existent.

IX. Discussion

In this section, we first highlight the key takeaways from our analysis of the data that Aristaeus collected, and then explore how the size of our infrastructure relates to the number of bots discovered. We close by discussing Aristaeus's limitations as well as future work directions.

A. Key Takeaways

- **Everyone is a target:** Just by being online and publicly accessible, each one of Aristaeus' honeysites attracted an average of 37,753 requests per month, 21,523 (57%) of which were clearly malicious. Each online site is exposed to fingerprinting and a wide range of attacks, abusing both operator error (such as, common passwords) as well as recently-released exploits.
- **Most generic bot requests are generated by rudimentary HTTP libraries:** Throughout our data analysis, we observed that 99.3% of the bots that visit our websites do not support JavaScript. This renders the state-of-the-art browser fingerprinting that is based on advanced browser APIs and JavaScript, ineffective. To combat this, we demonstrated that TLS fingerprinting can be used to accurately fingerprint browsing environments based on common HTTP libraries.
- **Most bots are located in residential IP space:** Through our experiments, we observed that the majority (64.37%) of bot IP addresses were residential ones, while only 30.36% of IP addresses were located in data centers. This indicates that bots use infected or otherwise proxied residential devices to scan the Internet. We expect that requests from residential IP space are less susceptible to rate limiting and blocklisting compared to requests from data centers and public clouds, out of fear of blocking residential users.
- **Generic bots target low-hanging fruit:** Aristaeus' logs reveal that 89.5% of sessions include less than 20 requests, and less than 0.13% of sessions include over 1,000 requests. The bruteforce attempts exhibit similar patterns: 99.6% IP addresses issue fewer than 10 attempts per domain, while only 0.3% IP addresses issued more than 100 attempts per domain. This indicates that most bots are highly selective and surgical in their attacks, going after easy-to-exploit targets.
- **IP blocklists are incomplete:** The vast majority (87%) of the malicious bot IPs from our honeysite logs were not listed in popular IP blocklists. This further emphasizes the limited benefits of static IP blocklisting and therefore the need for reactive defenses against bots. At the same time, the poor blocklist coverage showcases the practical benefits of Aristaeus, which can discover tens of thousands of malicious clients that are currently missing from popular blocklists.

- **Exploits that go public are quickly abused:** We observed that bots start probing for vulnerable targets as quickly as on the same day that an exploit was made public. Our findings highlight the importance of *immediately* updating Internet-facing software, as any delay, however small, can be capitalized by automated bots to compromise systems and services.
- **Fake search-engine bots are not as common as one would expect:** Contrary to our expectations, less than 0.3% of the total requests that claimed to be a search-engine bot were lying about their identity. This could suggest that either search-engine bots do not receive special treatments by websites, or that provided mechanisms for verifying the source IP address of search-engine bots have been successful in deterring bot authors from pretending to be search engines.
- **Most generic internet bots use the same underlying HTTP libraries:** Even though Aristaeus recorded more than 10.2 million HTTPS requests from bots, these bots generated just 558 unique TLS fingerprints. We were able to trace these fingerprints back to 14 popular tools and HTTP libraries, responsible for more than 97.2% of the total requests.

B. *Size of Infrastructure*

To understand how the number of Aristaeus-managed honeysites affects the collected intelligence (e.g. could Aristaeus record just as many attackers with 50 instead of 100 honeysites), we conduct the following simulation. We pick an ordering of honeysites at random and calculate the number of unique IP addresses and TLS fingerprints, each honeysite contributes. Figure 7 shows the results when this simulation is repeated 100 times. We resort to simulation (instead of just ordering the honeysites by contributing intelligence) since a new real deployment would not have access to post-deployment statistics.

We observe two clearly different distributions. While one could obtain approximately 50% of the TLS fingerprints with just 10% of the honeysites, the number of unique IP addresses grows linearly with the number of honeysites. Our findings indicate that, if the objective is the curation of TLS fingeprints from bots, a smaller infrastructure can suffice yet, if the objective is the collection of bot IP addresses, one could deploy an even larger number of honeysites and still obtain new observations.

C. *Limitations and Future Work*

In this study, we deployed honeysites based on five popular web applications. Since we discovered that many bots first fingerprint their targets before sending exploits (Section VI-A), our honeysites will not always capture exploit attempts towards unsupported web applications.

By design, Aristaeus in general and honeysites in particular, attract traffic from generic bots that target every website on the Internet. Yet high-profile websites as well as specific companies, often receive targeted bot attacks that are tailored to their infrastructure. These bots and their attacks will likely not be captured by Aristaeus, unless attackers first tried them on the public web.

As follow-up work, we plan to design honeysites that can dynamically react to bots, deceiving them into believing that they are interacting with the web application that they are looking for. Malware analysis systems already make use of variations of this technique to, e.g., detect malicious browser extensions [58] and malicious websites that attempt to compromise vulnerable browser plugins [59]. In this way, we expect that our honeysites will be able to capture more attacks (particularly from the single-shot scanners that currently do not send any traffic past an initial probing request) and exploit payloads.

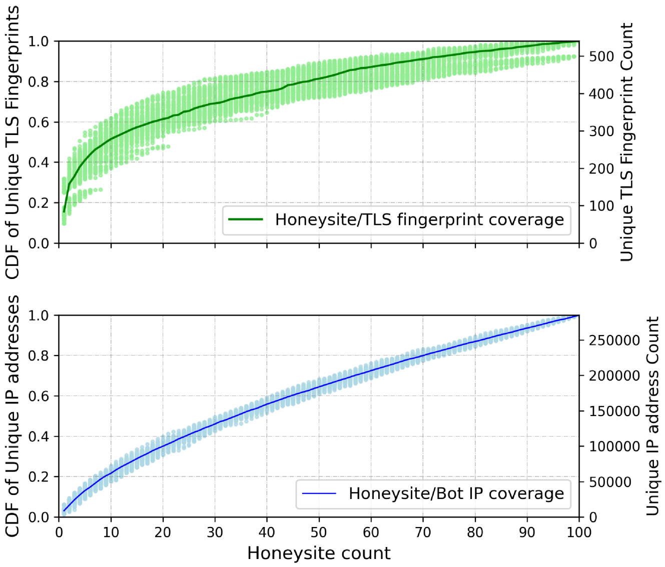

Fig. 7: *(Top) Number of honeysites and their coverage of bot TLS fingerprints. The opaque line is the median of 100 random experiments. (Bottom) Number of honeysites and their coverage of bot IP addresses.*

X. Related Work

Characterization and detection of crawlers. To quantify crawler behavior and differences of crawler strategies, a number of researchers analyzed large corpora of web server logs and reported on their findings. These studies include the analysis of 24 hours worth of crawler traffic on microsoft.com [60], 12 weeks of traffic from the website belonging to the Standard Performance Evaluation Corporation (SPEC) [61], 12.5 years of network traffic logs collected at the Lawrence Berkeley National Laboratory [62], as well as crawler traffic on academic networks [63] and publication libraries [64].

The security aspect of web crawlers has received significantly less attention compared to studies of generic crawling activity. Most existing attempts to differentiate crawlers from real users use differences in their navigational patterns, such as, the percentage of HTTP methods in requests (GET/POST/HEAD etc.), the types of links requested, and the timing between individual requests [4]–[6]. These features are then used in one or more supervised machine-learning algorithms trained using ground truth that the authors of each paper were able to procure, typically by manually labeling traffic of one or more webservers to which they had access. Xie et al. propose an offline method for identifying malicious crawlers by searching for clusters of requests towards non-existent resources [65]. Park et al. [22] investigated the possibility of detecting malicious web crawlers by looking for mouse movement, the loading of Cascading Style Sheets (CSS), and the following of invisible links. Jan et al. [66] propose an ML-based bot detection scheme trained on limited labeled bot traffic from an industry partner. By using data augmentation methods, they are able to synthesize samples and expand their dataset used to train ML models.

In our paper, we sidestep the issue of having to manually label traffic as belonging to malicious crawlers, through the use of honeysites, *i.e.,* websites which are never advertised and therefore any visitors must, by definition, either be benign/malicious crawlers or their operators who follow-up on a discovery from one of their bots. We therefore argue that the access logs that we collected via our network of honeysites can be used as ground truth in order to train more accurate ML-based, bot detection systems.

Network telescopes and honeypots. Network telescopes and honeypots are two classes of systems developed to study Internet scanning activity at scale. First introduced by Moore *et al.* [67], Network Telescopes observe and measure the remote network security events by using a customized router to reroute invalid IP address blocks traffic to a collection server. These works are effective in capturing network security events such as DDoS attacks, Internet scanners, or worm infections. Recent work by Richter *et al.* [68], applies the same principles to 89,000 CDN servers spread across 172 Class A prefixes and find that 32% all logged scan traffic are the result of localized scans. While large in scale, network telescopes either do not provide any response or interactive actions, or support basic responses at the network level (*e.g.,* sending back SYN-ACK if received a SYN from certain ports [69]). This makes them incapable of analyzing crawler interaction with servers and web applications.

Honeypots provide decoy computing resources for the purpose of monitoring and logging the activities of entities that probe them. High-interaction honeypots can respond to probes with high fidelity, but are hard to set up and maintain. In contrast, low-interaction honeypots such as Honeyd [70] and SGNET [71] intercept traffic sent to nonexistent hosts and use simulated systems with various "personalities" to form responses. This allows low-interaction honeypots to be somewhat extensible while limiting their ability to respond to the probes [72].

Our system, Aristaeus, combines some of the best properties of both these worlds. Aristaeus is extensible and can be automatically deployed on globally-dispersed servers. However, unlike network telescopes and low-interactive honeypots, our system does not restrict itself to the network layer responses but instead utilizes real web applications that have been augmented to perform traditional as well as novel types of client fingerprinting. In these aspects, our work most closely relates to the honeynet system by Canali and Balzarotti which utilized 500 honeypot websites with known vulnerabilities (such as SQL injections and Remote Command Execution bugs), and studied the exploitation and post-exploitation behavior of attackers [12]. While our systems share similarities (such as the use of real web applications instead of mock web applications or low-interaction webserver honeypots), our focus is on characterizing the requests that we receive, clustering them into crawling campaigns, and uncovering the real identity of crawlers. In contrast, because of their setup, Canali and Balzarotti [12] are able to characterize how exactly attackers attempt to exploit known vulnerabilities, what types of files they upload to the compromised servers, and how attackers abuse the compromised servers for phishing and spamming (all of which are well outside Aristaeus's goals and capabilities).

XI. Conclusion

In this paper, we presented the design and implementation of Aristaeus, a system for deploying and managing large numbers of web applications which are deployed on previously-unused domain names, for the sole purpose of attracting web bots. Using Aristaeus, we conducted a seven-month-long, large-scale study of crawling activity recorded at 100 globally distributed honeysites. These honeysites captured more than 200 GB of crawling activity, on websites that have *zero organic traffic*.

By analyzing this data, we discovered not only the expected bots operated by search engines, but an active and diverse ecosystem of malicious bots that constantly probed our infrastructure for operator errors (such as poor credentials and sensitive files) as well as vulnerable versions of online software. Among others, we discovered that an average Aristaeus-managed honeysite received more than 37K requests per month from bots, 50% of which were malicious. Out of the 76,000 IP addresses operated by clearly malicious bots recorded by Aristaeus, 87% of them are currently missing from popular IP-based blocklists. We observed that malicious bots engage in brute-force attacks, web application fingerprinting, and can rapidly add new exploits to their abusive capabilities, even on the same day as an exploit becoming public. Finally, through novel header-based, TLS-based, and JavaScript-based fingerprinting techniques, we uncovered the true identity of bots finding that most bots that claim to be a popular browser are in fact lying and are instead implemented on simple HTTP libraries built using Python and Go.

Next to all the insights into the abuse by malicious bots, Aristaeus allowed us to curate a dataset that is virtually free of organic user traffic which we will make available to researchers upon publication of this paper. This bot-only dataset can be used to better understand the dynamics of bots and design more accurate bot-detection algorithms.

XII. Availability

One of the main contributions of this paper is the curation of a bot-only, traffic dataset. To facilitate and advance research on the topic of bot detection, our dataset will be available to other researchers upon request.

Acknowledgment

We thank the reviewers for their valuable feedback. This work was supported by the Office of Naval Research under grant N00014-20-1-2720, N00014-20-1-2858, by the National Science Foundation under grants CNS-1813974, CNS-1941617, and CMMI-1842020, as well as by a 2018 Amazon Research award. Any opinions, findings, or conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] A. Shirokova, "Cms brute force attacks are still a threat." [Online]. Available: https://blogs.cisco.com/security/cms-brute-force-attacks-are-still-a-threat

[2] T. Canavan, *CMS Security Handbook: The Comprehensive Guide for WordPress, Joomla, Drupal, and Plone*. John Wiley and Sons, 2011.

[3] Imperva, "Bad bot report 2020: Bad bots strike back." [Online]. Available: https://www.imperva.com/resources/resource-library/reports/2020-bad-bot-report/

[4] A. G. Lourenço and O. O. Belo, "Catching web crawlers in the act," in *Proceedings of the 6th international Conference on Web Engineering*, 2006, pp. 265–272.

[5] P.-N. Tan and V. Kumar, "Discovery of web robot sessions based on their navigational patterns," in *Intelligent Technologies for Information Analysis*. Springer, 2004, pp. 193–222.

[6] G. Jacob, E. Kirda, C. Kruegel, and G. Vigna, "Pubcrawl: Protecting users and businesses from crawlers," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, 2012, pp. 507–522.

[7] A. Vastel, W. Rudametkin, R. Rouvoy, and X. Blanc, "FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers," in *MADWeb'20 - NDSS Workshop on Measurements, Attacks, and Defenses for the Web*.

[8] B. Amin Azad, O. Starov, P. Laperdrix, and N. Nikiforakis, "Web Runner 2049: Evaluating Third-Party Anti-bot Services," in *17th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02612454

[9] K. Bock, D. Patel, G. Hughey, and D. Levin, "uncaptcha: a low-resource defeat of recaptcha's audio challenge," in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[10] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage, "Re: Captchas-understanding captcha-solving services in an economic context." in *USENIX Security Symposium*, vol. 10, 2010, p. 3.

[11] S. Sivakorn, J. Polakis, and A. D. Keromytis, "I'm not a human: Breaking the google recaptcha," *Black Hat*, 2016.

[12] D. Canali and D. Balzarotti, "Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web," in *Proceedidngs of the 20th Network & Distributed System Security Symposium (NDSS)*, 2013.

[13] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis, "Domain-z: 28 registrations later measuring the exploitation of residual trust in domains," in *IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 691–706.

[14] "Seleniumhq browser automation," https://www.selenium.dev/.

[15] P. Eckersley, "How unique is your web browser?" in *International Symposium on Privacy Enhancing Technologies Symposium*, 2010, pp. 1–18.

[16] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 541–555.

[17] O. Starov and N. Nikiforakis, "Xhound: Quantifying the fingerprintability of browser extensions," in *IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 941–956.

[18] P. Laperdrix, W. Rudametkin, and B. Baudry, "Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints," in *IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 878–894.

[19] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, E. Weippl, and F. Wien, "Fast and reliable browser identification with javascript engine fingerprinting," in *Web 2.0 Workshop on Security and Privacy (W2SP)*, vol. 5, 2013.

[20] L. Brotherston, "Tls fingerprinting." [Online]. Available: https://github.com/LeeBrotherston/tls-fingerprinting

[21] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of https interception." in *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS)*, 2017.

[22] K. Park, V. S. Pai, K.-W. Lee, and S. B. Calo, "Securing web service by automatic robot detection." in *USENIX Annual Technical Conference, General Track*, 2006, pp. 255–260.

[23] G. Analytics, "How a web session is defined in analytics." [Online]. Available: https://support.google.com/analytics/answer/2731565?hl=en

[24] Valve, "Fingerprintjs2." [Online]. Available: https://github.com/Valve/fingerprintjs2

[25] M. W. Docs, "Content security policy (csp)." [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[26] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 921–930.

[27] N. Virvilis, B. Vanautgaerden, and O. S. Serrano, "Changing the game: The art of deceiving sophisticated attackers," in *2014 6th International Conference On Cyber Conflict (CyCon 2014)*. IEEE, 2014, pp. 87–97.

[28] B. P. Ltd, "Web technology usage trends (accessed march 27,2020)." [Online]. Available: https://trends.builtwith.com

[29] "Wordpress: About us," https://wordpress.com/about/.

[30] "Ip2location lite ip-asn database," https://lite.ip2location.com/database/ip-asn.

[31] X. Mi, X. Feng, X. Liao, B. Liu, X. Wang, F. Qian, Z. Li, S. Alrwais, L. Sun, and Y. Liu, "Resident evil: Understanding residential ip proxy as a dark service," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[32] D. Liu, S. Hao, and H. Wang, "All your dns records point to us: Understanding the security threats of dangling dns records," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1414–1425.

[33] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna, "Cloud strife: mitigating the security risks of domain-validated certificates," 2018.

[34] S. F. McKenna, "Detection and classification of web robots with honeypots," Naval Postgraduate School Monterey United States, Tech. Rep., 2016.

[35] R. Barnett, "Setting HoneyTraps with ModSecurity: Adding Fake robots.txt Disallow Entries," https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/setting-honeytraps-with-modsecurity-adding-fake-robotstxt-disallow-entries/.

[36] R. Haswell, "Stopping bad robots with honeytraps," https://www.davidnaylor.co.uk/stopping-bad-robots-with-honeytraps.html.

[37] Google, "Verifying googlebot." [Online]. Available: https://support.google.com/webmasters/answer/80553

[38] Microsoft, "Verifying bingbot." [Online]. Available: https://www.bing.com/toolbox/verify-bingbot

[39] Yandex, "Verifying yandexbot." [Online]. Available: https://yandex.com/support/webmaster/robot-workings/check-yandex-robots.html

[40] Baidu, "How can i know the crawling is from baiduspider." [Online]. Available: https://help.baidu.com/question?prod_id=99&class=0&id=3001

[41] Google, "Overview of google crawlers." [Online]. Available: https://support.google.com/webmasters/answer/1061943

[42] Netcraft, "Netcraft: Active cyber defence," *URL: https://www.netcraft.com/*, 2014.

[43] "Internet Archive: Digital Library of Free & Borrowable Books, Movies, Music & Wayback Machine," https://archive.org/.

[44] C. Security, "Multiple vulnerabilities in draytek products could allow for arbitrary code execution." [Online]. Available: https://www.cisecurity.org/advisory/multiple-vulnerabilities-in-draytek-products-could-allow-for-arbitrary-code-execution_2020-043/

[45] M. Daniel, H. Jason, and g0tmi1k, "10k most common credentials." [Online]. Available: https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10k-most-common.txt

[46] "The blindelephant web application fingerprinter." [Online]. Available: https://github.com/lokifer/BlindElephant

[47] "Whatweb." [Online]. Available: https://github.com/urbanadventurer/WhatWeb

[48] "Thinkphp remote code execution vulnerability used to deploy malware." [Online]. Available: https://www.tenable.com/blog/thinkphp-remote-code-execution-vulnerability-used-to-deploy-variety-of-malware-cve-2018-20062

[49] "Exploit code published for two dangerous apache solr remote code execution flaws." [Online]. Available: https://www.zdnet.com/article/exploit-code-published-for-two-dangerous-apache-solr-remote-code-execution-flaws/

[50] O. Starov, J. Dahse, S. S. Ahmad, T. Holz, and N. Nikiforakis, "No honor among thieves: A large-scale analysis of malicious web shells," in *Proceedings of the 25th International Conference on World Wide Web*, ser. WWW '16, 2016, p. 1021–1032.

[51] "Seclists: A collection of multiple types of lists used during security assessments." [Online]. Available: https://github.com/danielmiessler/SecLists

[52] "lhlsec/webshell." [Online]. Available: https://github.com/lhlsec/webshell

[53] R. D. Graham, "Masscan: Mass ip port scanner," *URL: https://github.com/robertdavidgraham/masscan*, 2014.

[54] Z. Durumeric, E. Wustrow, and J. A. Halderman, "Zmap: Fast internet-wide scanning and its security applications," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 605–620.

[55] D. B. Internet Engineering Task Force (IETF), "Applying generate random extensions and sustain extensibility (grease) to tls extensibility." [Online]. Available: https://tools.ietf.org/html/rfc8701

[56] "Chrome platform status: Grease for tls," https://www.chromestatus.com/feature/6475903378915328.

[57] "Cve-2020-0618: Rce in sql server reporting services (ssrs)." [Online]. Available: https://www.mdsec.co.uk/2020/02/cve-2020-0618-rce-in-sql-server-reporting-services-ssrs/

[58] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson, "Hulk: Eliciting malicious behavior in browser extensions," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 641–654.

[59] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious javascript code," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 281–290.

[60] J. Lee, S. Cha, D. Lee, and H. Lee, "Classification of web robots: An empirical study based on over one billion requests," *computers & security*, vol. 28, no. 8, pp. 795–802, 2009.

[61] M. C. Calzarossa, L. Massari, and D. Tessera, "An extensive study of web robots traffic," in *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, 2013, p. 410.

[62] M. Allman, V. Paxson, and J. Terrell, "A brief history of scanning," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 77–82.

[63] M. D. Dikaiakos, A. Stassopoulou, and L. Papageorgiou, "An investigation of web crawler behavior: characterization and metrics," *Computer Communications*, vol. 28, no. 8, pp. 880–897, 2005.

[64] P. Huntington, D. Nicholas, and H. R. Jamali, "Website usage metrics: A re-assessment of session data," *Information Processing & Management*, vol. 44, no. 1, pp. 358–372, 2008.

[65] G. Xie, H. Hang, and M. Faloutsos, "Scanner hunter: Understanding http scanning traffic," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 27–38.

[66] S. T. Jan, Q. Hao, T. Hu, J. Pu, S. Oswal, G. Wang, and B. Viswanath, "Throwing darts in the dark? detecting bots with limited data using neural data augmentation," *The 41st IEEE Symposium on Security and Privacy (IEEE SP)*, Jan 2020.

[67] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Network telescopes: Technical report," Cooperative Association for Internet Data Analysis (CAIDA), Tech. Rep., 2004.

[68] P. Richter and A. Berger, "Scanning the scanners: Sensing the internet from a massively distributed network telescope," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 144–157.

[69] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The internet motion sensor-a distributed blackhole monitoring system." in *NDSS*, 2005.

[70] N. Provos, "A virtual honeypot framework." in *USENIX Security Symposium*, vol. 173, 2004, pp. 1–14.

[71] C. Leita and M. Dacier, "Sgnet: a worldwide deployable framework to support the analysis of malware threat models," in *2008 Seventh European Dependable Computing Conference*. IEEE, 2008, pp. 99–109.

[72] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," *ACM SIGCOMM computer communication review*, vol. 34, no. 1, pp. 51–56, 2004.

XIII. Appendix

TABLE VI: *Aristaeus dataset description*

Dataset	Requests	%Requests	Unique IP Addresses	Blocklist Coverage	Shared Crawling	Begin Date	End Date
Benign	347,386	1.3%	6,802	6.91%	Yes	2020-01-24 00:00:01	2020-08-24 23:59:59
Malicious	15,064,878	57%	76,396	13%	No		
Unknown / Gray	11,015,403	41.68%	206,111	11.64%	No		
Total	26.4 million	100%	287K	11.61%	(Mixed)		

TABLE VII: *Top requested URL in different web applications*

Rank	WordPress	Joomla	Drupal	PHPMyAdmin	Webmin
1	/xmlrpc.php (62.664%)	/administrator-/index.php (48.333%)	/user/login?destination=/node/1#comment-form (81.143%)	(POST)/index.php (75.65%)	/session_login.cgi (79.93%)
2	/wp-login.php (25.094%)	/administrator/ (41.512%)	/wp-login.php (18.064%)	(POST)/phpmyadmin/index.php (9.658%)	/wp-login.php (13.649%)
3	/wp-admin/ (12.239%)	/wp-login.php (9.29%)	/xmlrpc.php (0.476%)	(GET)/phpmyadmin/index.php (5.715%)	/xmlrpc.php (4.51%)
4	/administrator/index.php (0.001%)	/xmlrpc.php (0.822%)	/administrator/ (0.222%)	/wp-login.php (8.228%)	/robots.txt (1.684%)
5	/administrator (0.001%)	/wp-admin/ (0.044%)	/administrator/index.php (0.095%)	/vendor/phpunit/phpunit/src/Util/PHP/eval-stdin.php (0.749%)	/wp-admin/ (0.227%)