# Empirical Scanning Analysis of Censys and Shodan

Christopher Bennett, AbdelRahman Abdou, and Paul C. van Oorschot
School of Computer Science, Carleton University, Canada

***Abstract*—Engines that scan Internet-connected devices allow for fast retrieval of useful information regarding said devices, and their running services. Examples of such engines include Censys and Shodan. We present a snapshot of our in-progress effort towards the characterization and systematic evaluation of such engines, herein focusing on results obtained from an empirical study that sheds light on several aspects. These include: the freshness of a result obtained from querying Censys and Shodan, the resources they consume from the scanned devices, and several interesting operational differences between engines observed from the network edge. Preliminary results confirm that the information retrieved from both engines can reflect updates within 24 hours, which aligns with implicit usage expectations in recent literature. The results also suggest that the consumed resources appear insignificant for common Internet applications, *e.g.,* one full application-layer connection (banner grab) per port, per day. Results so far highlight the value of such engines to the research community.**

## I. Introduction

As the Internet evolves, and its usage morphs over time, the need for systemic search engines that help us understand more about its connected devices arose. Censys [1] and Shodan [2] are two prominent such engines, increasingly relied upon recently by researchers and practitioners alike. Censys was realized in 2015 [13], and is designed to increase the accessibility of Internet-wide scanning by providing an approximate real-time view of the Internet as an online, publicly available, service. Shodan was realized in 2009 as a commercial service, and was targeted to provide real-time "market intelligence" about Internet-connected devices [20].

Censys is built upon a suite of technologies (*e.g.,* ZMap [14], ZGrab [3]) that provide the means to efficiently enumerate the IPv4 address space and contact devices therein. It first scans the entire IPv4 address space for a single port (*i.e.,* horizontal scanning), then further collects *banners* (*i.e.,* metadata about a service, or the resources served by such a service) from responding ports. Shodan, on the other hand, is less transparent; how it conducts Internet-wide scans remains unclear to the public.

From the perspective of the scanned devices, both Censys and Shodan provide means to allow administrators to opt-out from scans, thus abiding by widely agreed-upon Internet citizenship habits [14]. This is analogous to the means available to website owners to signal to a web search engine (like Google and Bing) their lack of desire to have their website indexed (*e.g.,* using the `robots.txt` file). Both Censys and Shodan provide a search interface on their websites, where a user may enter a query and retrieve a list of matching IP addresses; and both return only a subset of the results for free, and require a payment for the full set.

Upon conducting a preliminary empirical analysis, we provide insights into the following questions:

- How fresh are the results obtained from searching Censys/Shodan (Sec. III)?
- How much resources can we expect to be consumed from the scanned devices by Censys/Shodan scans (Sec. IV)?
- From the network edge, how does the operation of Censys and Shodan differ (Sec. V)?

Approaching these question can help validate research relying on these engines. It can also provide insights to: aid practitioners in deciding whether to opt-out from scans, understand the context from which results to their search queries are returned, and spark longer-term discussions regarding the value (relative to the cost) of such large-scale scans.

We setup virtual machines (VMs) in five locations: San Jose (California), Tokyo, Montreal, Paris, and Sao Paulo, and ran four popular services on each VM to receive port scans and banner grabs. This allows us to receive traffic from Censys and Shodan, and analyze and compare scanning patterns, thus approach the above questions. We now detail our data collection methodology.

## II. Data Collection and Analysis Methodology

We used Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances to host our VMs. Each EC2 instance ran Linux 2 Amazon Machine Images (AMI) on top of Linux kernel 4.14 64bit(x86). VMs were assigned a public IP address that remained unchanged for the duration of our experiment. Each VM ran four public-facing services: FTP (using `vsftpd`), SSH (using `Open SSH`), HTTP and HTTPS (using `Apache`), thus having four open ports, respectively 21, 22, 80, and 443. Firewalls were configured with no restrictions, allowing any traffic inbound and outbound. Data collection lasted 47 days, beginning on March 10, 2020.

We used the `tcpdump` packet capture tool on each VM to collect all network traffic as `pcap` (packet capture) files, which were later searched for Censys and Shodan scans.

**Identifying Censys' scans.** Censys' IP address are publicly known; during our experiment, Censys was scanning from the subnet `198.108.66.0/23` [4]. We assume that any packet whose source IP address matches this subnet is from Censys. We saw 186 IP addresses from this subnet.

**Identifying Shodan's scans.** Unlike Censys, Shodan does not announce its source IP addresses. To identify them, we

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2021
25 February 2021
ISBN 1-891562-67-3
https://dx.doi.org/10.14722/madweb.2021.23xxx
www.ndss-symposium.org

ran a DNS history search for the domain `*.shodan.io` using SecurityTrails [5]. The SeucrityTrails API allowed us to find the first time it became aware of a domain (*first seen*) and when it became aware of the domain changing (*last seen*). Any packet with source IP address belonging to one of Shodan's subdomains is assumed to be from Shodan if the timestamp on the first packet in the TCP session was after the *first seen* date and before *last seen*. Following this methodology, we saw 16 IP addresses with corresponding names matching `*.shodan.io`. To confirm that these belong to independent Shodan scanners, we compared their SHA1 hashes to Shodan's *crawler* field—a 40 character hexadecimal string returned upon querying Shodan. Each of the 16 hashes appeared in the *crawler* field at least once, suggesting that a single IP address likely correlates to a single scanner.

**Classifying sessions into Port Scans and Banner Grabs.** We analyze port scans and banner grabs independently. To do so, some processing needs to be made on the `pcap` files because, (1) `pcap` files are raw, containing one packet per line, so the representation of a "TCP session" is not directly available; (2) we need a consistent methodology to classify several possible cases of the received packets—cases such as receiving only a TCP SYN packet, only a RST packet, or a SYN followed by a SYN/ACK and RST; and (3) we configured five open ports on each VM, but we still received scans on other (closed) ports. We used the Python library `pcap-splitter` [6] to extract and group together packets that belong to the same TCP session. We then categorized each session (grouped packets) as follows:

- *S*: A session of a single packet having the SYN flag set.
- *SRE*: First packet has SYN flag, second has RST.
- *SR*: First packet has only SYN flag, second has RST.
- *SAR*: First packet has only SYN flag, second has only SYN and ACK flags, last has RST.
- *BG*: Any TCP session that reaches the TCP ESTABLISHED state [19], contains at least one packet with an ACK value $>1$, and ends with a TCP teardown or RST.
- *Dangling TCP Handshake*: Any TCP session that does not contain a packet with an ACK value $>1$, and contains no packets with RST or FIN flags.
- *Dangling TCP Session*: Any TCP session that contains a packet with an ACK value $>1$, and contains no packets with RST or FIN flags.

We now formally distinguish a *Closed Port Scan* from an *Open Port Scan* from a *Banner Grab* as follows:

– *Closed Port Scan* = *SR* $\cup$ *SRE*
– *Open Port Scan* = *SAR* $\cup$ *Dangling TCP Handshake*
– *Banner Grab* = *BG* $\cup$ *Dangling TCP Session*

## III. Freshness of Search Results

We analyze the freshness of the results returned upon running a search query. For example, if a banner search returns 1.2.3.4:80, is this information stale? In other terms, if the banner is no longer served by this device, how long would it take Censys/Shodan to capture this? And how long would it take them to reflect the change in their search results?

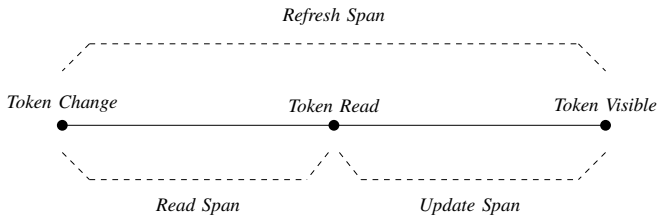



Fig. 1: Variables used to assess the freshness of search results.

Herein we only focus on the freshness of banner grabs on port 80 (HTTP). To analyze this, we generated a unique 64 character string (*token* henceforth) for each VM everyday, for the duration of our data collection phase (47 days). We updated the `index.html` page everyday at 10pm EST to serve the new token. We call this the *Token Change* instance. We then searched our `pcaps` for the moment when an HTTP GET response (with the new token) was returned to a banner grab from Censys/Shodan—the *Token Read* instance. Finally, everyday we used Censys' and Shodan's API search interfaces to find the new token. If found, we call this the *Token Visible* instance. We then analyze the time delays between the three instances, namely: *Read Span*, *Update Span*, and *Refresh Span* (Fig. 1). The *Refresh Span* chart is particularly helpful as it shows the delay between the changed event and the reflection of this change to the user. Note that because it is a function of the time when we queried both engines for the token, the *Token Visible* instance is an upper bound, *i.e.,* the token could have become visible sooner. Accordingly, this applies to the *Refresh Span* and *Update Span* delay variables.

By inspecting our `pcap` files, we found that many tokens were never captured (*i.e.,* banner grabbed) by Censys and Shodan. That is, our next token would replace the previous one before any banner grab occurs on the replaced token. In fact, we found that out of the 47 (days) $\times$ 5 (VMs) $=$ 235 tokens we generated, only 25 were captured by Censys and 19 by Shodan. More surprising is the fact that for the duration of the 47 days, we received 46 HTTP banner grabs from Censys (*i.e.,* summing up all five VMs). From Shodan, we received 102 HTTP banner grabs. Neither Shodan nor Censys would have been able to capture all tokens, even if the banner grabs were spaced perfectly.

To analyze their efficiency in reflecting updates onto their search interfaces, we calculate how fast did the 25 Censys-captured and the 19 Shodan-captured tokens become visible after they were captured. Figure 2 shows CDFs of the three delay variables for the captured tokens. The results show that both engines are fast to refresh their query responses, with 65% of captured tokens becoming visible on their search interfaces (*Update Span*) within $\sim$15 hours of capturing. Shodan was slightly faster; 20% of its captured tokens became visible within 5 hours of being captured (15 hours for Censys). For the *Refresh Span*, both engines made all captured tokens visible within 24 hours of generating these tokens.

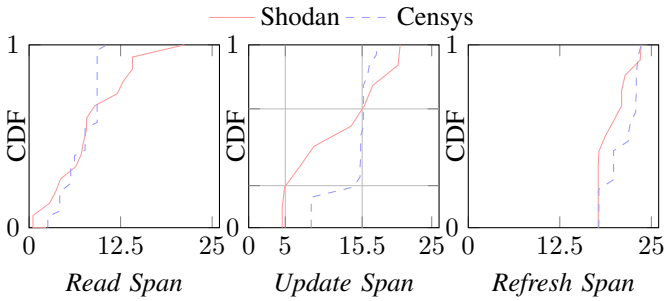



Fig. 2: CDFs of freshness results, in hours.

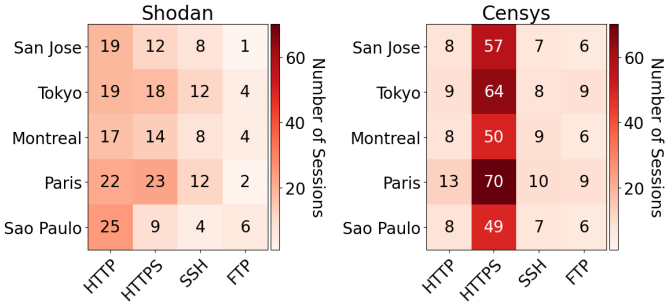



Fig. 4: Number of banner grabs for each service on each VM.

## IV. Scanning Resource Consumption

We now switch to resource consumption. In particular, using our collected `pcap` files, we measure the number of SYN scans and banner grabs per time window (Sec. IV-A), and the duration with which TCP sessions remain open (Sec. IV-B).

### A. Number of Scans

Over the course of our data collection phase (47 days), each VM received ∼27,614 SYN scans, or a little more than 587 scans per day (176 per day from Shodan, 411 from Censys). We focus on open port scans and banner grabs, as these generate more traffic than closed port scans. Figures 3 and 4 show heat maps of the number of open port scans and banner grabs for each service on each VM. For Censys, despite the darkness of the (vertical) HTTPS bar (*e.g.,* averaging more than five times that of HTTP), Censys has generated an average of ∼97 open port scans on each VM, Shodan half that. According to our results, a server with only ports 443 and 80 open can expect to receive $< 40$ banner grabs per month from each engine. These results are relatively consistent across our five VMs, with the exception of Paris and Tokyo receiving 37% and 24% more HTTPS scans from Censys than other locations. Sao Paulo received the largest number of HTTP scans from Shodan.

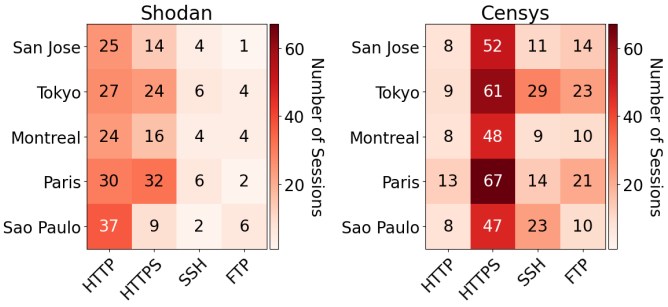



Fig. 3: Number of open port scans for each service per VM.

### B. Duration of Scans

A TCP session duration is the time difference between receiving the first TCP SYN packet (hence, resources allocated) and the event when the session was terminated (hence, resources released). The latter occurs when a TCP RST is received/sent, when the second TCP FIN packet is received/sent, or when a session times-out and gets terminated by the OS. This applies to closed/open port scans and banner grabs.

For closed ports, almost all scans finished in $<$ 1ms. For open port scans and banner grabs, Fig. 5 shows CDFs of the session duration. On average, open port scans lasted ∼1s (139ms Censys and over 2s Shodan). This can be acceptable in many cases in practice, depending on the available resources and the number of open ports. Banner grab sessions took slightly longer, averaging $<$ 3s across all services and VMs. The longest session lasted ∼248s, which was an HTTPS banner grab from Shodan running on the Montreal VM; the banner grab was completed in 42ms, but the final RST[1] packet was delayed over 248s. The fastest closed port scan was from Shodan on port 10554 (on the San Jose VM), which was terminated almost instantly.

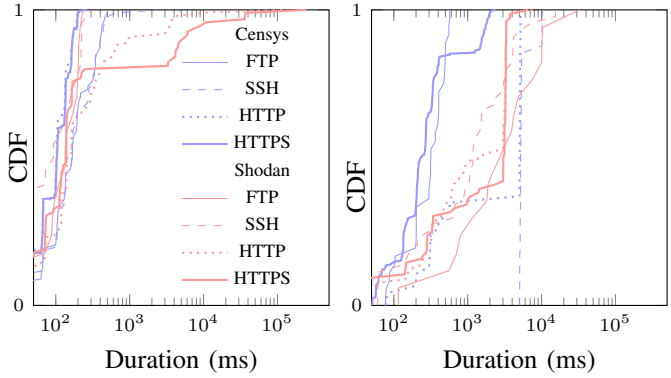



Fig. 5: Duration of open port scans (left) and banner grabs (right) received by all VMs on four services.

## V. Theory of Operation: Censys vs Shodan

We now present a preliminary comparison between Censys and Shodan, viewing their operation from the network edge. We again use information extracted from our five VMs and the `pcaps` collected. We focus our comparison on: the services both engines appear interested in (Sec. V-A), their available

[1]Most scans were terminated with a RST from Censys/Shodan, not a FIN.

resources, particularly their pool of IP addresses (Sec. V-B), and their scanning configuration (Sec. V-C).

### A. Interest in services

Table I shows the five most scanned ports on each VM, ordered from most (left) to least (right) scanned. Censys and Shodan were different in the most commonly scanned port: 443 for Censys and 80 for Shodan (with the exception of Paris, where Shodan scanned 443 a little more than 80). Port 443 came second for Shodan. In contrast, ports 21 (FTP) and 22 (SSH) came second (almost tied) for Censys, with the exception of Montreal where ports 9200, 2082, 8081, and 8088 were scanned more than ports 21 and 22. In a stark contrast between both engines, our logs show that Shodan's most scanned port (80) came in the ninth position for Censys. The complete set of ports and services that Censys and Shodan may have been scanning for, as well as the IANA-registered service for these ports can be found, respectively, in Tables V and VI in the appendix.

| VM | TCP Port (P) Count (C) Most Common → Least Common | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | P | C | P | C | P | C | P | C | P | C |
| Top five Shodan-scanned ports | | | | | | | | | | |
| San Jose | 80 | 25 | 443 | 14 | 444 | 12 | 1177 | 11 | 81 | 10 |
| Tokyo | 80 | 27 | 443 | 24 | 1177 | 17 | 7443 | 14 | 53 | 14 |
| Montreal | 80 | 24 | 443 | 16 | 1177 | 13 | 81 | 11 | 4782 | 11 |
| Paris | 443 | 32 | 80 | 30 | 81 | 19 | 444 | 16 | 2086 | 15 |
| Sao Paulo | 80 | 37 | 443 | 9 | 3460 | 12 | 82 | 11 | 6666 | 11 |
| Top five Censys-scanned ports | | | | | | | | | | |
| San Jose | 443 | 52 | 21 | 14 | 5900 | 14 | 9200 | 13 | 83 | 13 |
| Tokyo | 443 | 61 | 22 | 29 | 21 | 23 | 9200 | 19 | 5902 | 18 |
| Montreal | 443 | 48 | 9200 | 13 | 2082 | 12 | 8081 | 11 | 8088 | 11 |
| Paris | 443 | 67 | 21 | 21 | 9200 | 19 | 5902 | 19 | 8090 | 18 |
| Sao Paulo | 443 | 47 | 22 | 23 | 9200 | 13 | 2083 | 13 | 8081 | 13 |

TABLE I: Most scanned ports.

### B. IP addresses: Pool and Configuration Characteristics

Figure 6 shows the geographic locations of Shodan's 16 and Censys' 186 IP addresses that we observed in our `pcaps` (Sec. II). WhoisXMLAPI [7] was used for location data.

Shodan's 16 IP addresses originate from four countries, whereas Censys' 186 addresses appear to be only in Michigan, USA. Figure 7(a) shows the number of TCP sessions initiated from these locations. In terms of the distribution of scanning effort on these locations, Fig. 7(b) shows that 40% of Shodan's 16 scanners contacted all of our five VMs, and about a third contacted only one VM. This is in contrast to Censys, where the majority of the 186 scanners contacted all of our VMs. This reflects what appears to be a fundamental difference between the configuration of both engines' scanners: Censys conducting more horizontal scans (each scanner potentially scanning the entire IPv4 space), whereas for Shodan the IPv4 space appears divided among its scanners. The latter arrangement can be more efficient, and can potentially reduce placement of such IP addresses in globally-coordinated blacklists. Additionally, Wan *et al.* [21] found that a single origin scan will miss 4% of HTTP(S) and 16% of SSH traffic. They found that scanning from two origins reduces this to 1.7% of HTTP(S) traffic missed and three origins further reduces the miss rate to 1%. This appears to provide a coverage advantage to Shodan over Censys. Nonetheless, Censys' scanners and activities are relatively more transparent.

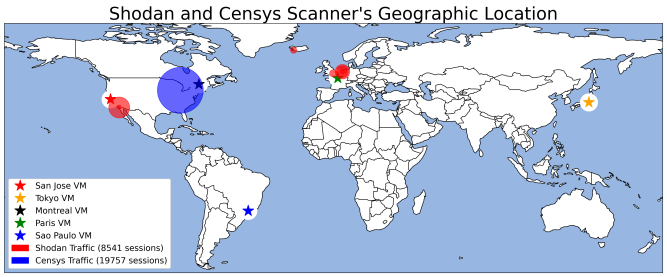
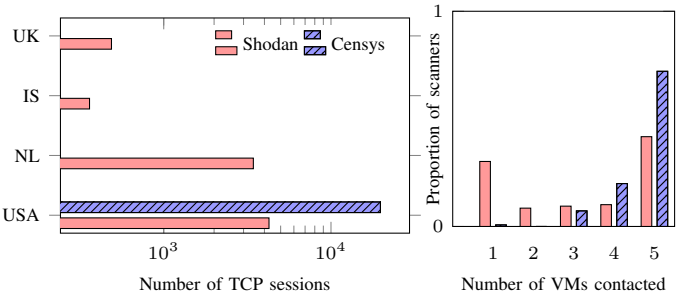



Fig. 6: Geographic locations of Censys (blue) and Shodan (red) scanners, and our VMs (stars). Marker size increases with the number of scans to our VMs. Map generated using Python Cartopy v0.18.0 [8].




(a) Scanning origins. (b) Distinctly contacted VMs

Fig. 7: Geographic targets and origins of scanners.

To test whether Shodan configures its geographically-scattered scanners to scan nearby devices, we calculate the average of the actual distances between Shodan's scanners and our VMs, $AD_v$, and the average of the distances between the endpoints of observed TCP session, $OD_v$, as follows:

$$AD_v = \sum_{s \in S} \frac{\textit{distance}(s, v)}{|S|}, \quad OD_v = \sum_{k \in K_v} \frac{\textit{distance}(F(k), v)}{|K_v|}$$

where $S$ is the set of Shodan scanners, $K_v$ is the set of TCP sessions involving VM $v$, $F(k)$ is the scanner that initiated TCP session $k$, and *distance*($x$,$y$) is the big circle geographic distance between points $x$ and $y$. A ratio of $AD_v$ to $OD_v$ close to 1 refutes the hypothesis that the geographic distance is a parameter in matching scanned devices with scanners. A ratio $\gg 1$ suggests that devices are contacted by nearby scanners, and $\ll 1$ suggests otherwise. Table II shows the results, which generally show no evidence of geographic bias, except for Montreal. Upon investigating further, we found that this is

due a nearby scanner being responsible for generating more Shodan traffic to all VMs than other Shodan scanners (some scanners only scan a subset of ports; see Sec. V-C below).

| VM | $AD_{vm}$ (km) | $OD_{vm}$ (km) | $\frac{AD_{vm}}{OD_{vm}}$ |
|---|---|---|---|
| San Jose | 4210 | 3645 | 1.15 |
| Tokyo | 9108 | 9950 | 0.9154 |
| Montreal | 4547 | 2009 | 2.2633 |
| Paris | 5304 | 5880 | 0.902 |
| Sao Paulo | 9769 | 8725 | 1.1197 |

TABLE II: Ratio of the average distance between Shodan scanners and each VM ($AD_{vm}$) to the average between the endpoints of observed TCP session ($OD_{vm}$).

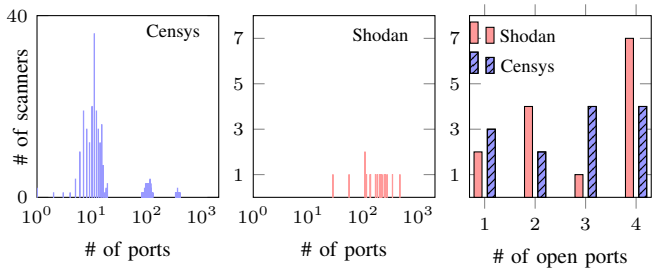



Fig. 8: Per scanner: number of ports scanned by Censys (left) and Shodan (middle), and the number of ports banner grabbed by both (right).

### C. Scanning Configuration

We shed light on several configuration parameters that we analyzed during our experiment. Specifically, analogous to the previous analysis regarding whether each scanner was responsible for scanning a dedicated subset of the IPv4 address space, we investigate whether every scanner has a dedicated set of port numbers (services) to scan on each target host. We also look at the means by which each engine is configured to request resources. Finally, we analyze and compare the timing consistency of the engines' scanning scripts.

**Port distribution across scanners.** We count the number of ports scanned by each scanner. For example, if a scanner made 1000 HTTP port scans, 1 HTTPS, and 10 SSH over the entire 47 days, then it has scanned three ports (namely, 443, 80, and 22). Figure 8 shows a histogram of the results. The majority of Censys scanners are focused on 20 or fewer ports, and very few scanners are assigned hundreds of ports to scan. Shodan appears to operate differently yet again, where each scanner is assigned a different number of ports, but generally in larger numbers than Censys. For example, only two of Shodan's scanners were assigned fewer than 200 ports; the rest were assigned between 200 and 1000 ports. For banner grabs, the chart on the right (Fig. 8) suggests that banner grabbing might be the responsibility of few Censys scanners—only 13 (out of 186) conducted banner grabbing. In contrast, most of Shodan's scanners conducted banner grabbing, with seven scanners (*i.e.,* half of Shodan's) banner-grabbed from all four open ports.

**Configuration of requesting resources.** We examine the HTTP GET requests in the banner grabs of Censys and Shodan to see what resources are being requested from our HTTP service, and how. Combing all five VMs, Censys used one user agent in all 46 HTTP banner grabs, and always requested the root path. Shodan on the other hand used several user agents (see Table III in the appendix for a complete list) and requested multiple paths (Table IV in the appendix). Using different user agents can provide a higher likelihood of bypassing traffic filters on the application layer. Configuring the scans to look as if they are coming from a browser can further enhance the accessibility of content; *e.g.,* many sites require a browser to have javascript enabled. The browser of choice was different across both engines, with Censys using "Mozilla/5.0 zgrab/0.x", and Shodan claiming to be a Google Chrome running on a Windows machine.

**Scanning consistency.** We examine and compare the consistency, in terms of the time of day, across both engines. Figures 9 and 10 show the standard deviation of hours between the start of each SYN scan and banner grab respectively. We require at least two events to calculate the time between them (hence, the standard deviation). The empty cells in Fig. 9 and 10 are due to having a single corresponding event. Overall, Shodan appears slightly more consistent than Censys.

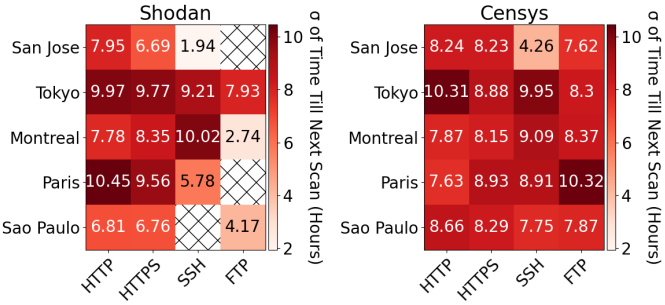



Fig. 9: The standard deviation ($\sigma$) of the number of hours between the start of each SYN scan.

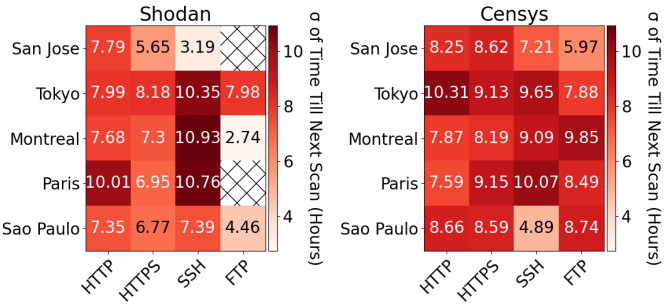



Fig. 10: The standard deviation ($\sigma$) of the number of hours between the start of each banner grab.

## VI. Related Work

**Identifying Internet-wide scanners.** Analogous to our methodology, Chen *et al.* [12] deployed six honeypots and collected three months worth of network traffic. They used machine learning on the scanning patterns, which found 16 scanners similar to Shodan but did not belong to Shodan. Upon analyzing patterns of other scanners (10 belong to Censys and 6 to PLCscan), and the services they most commonly targeted, Durumeric *et al.* [18] found that many focus on services commonly associated with vulnerabilities. Heo *et al.* [17] study network scanning trends from a 31-day-long connection log they obtained from two firewalls of a campus network. They analyzed over 21 billion combined TCP and UDP connections to determine the characteristics of network scans targeting the campus. They created heuristics to identify scanners (scanners were identified by their IP addresses, as we do herein), and to classify the type of scanning as horizontal or vertical.

**Evaluating search engine functionality.** In 2014, Bodenheim *et al.* [11] investigated the functionality of Shodan. They deployed, in one subnet, four Internet-connected Allen-Bradley ControlLogix Programmable Logic Controllers (PLCs) with a static IP address publicly reachable via the Internet. Each PLC exposed two services: HTTP (port 80) and a common industrial control service (port 44818). Two PLCs were left with the default configuration, one had its banner obfuscated, and one had its banner changed to claim it was an Allen-Bradly ControlLogix. All four PLCs received at least one port scan from Shodan in fewer than 4 days. (Shodan does not index port 44818.) The HTTP banners were grabbed from all four PLCs within 14 days. Using keywords only, the authors [11] searched Shodan for the PLCs; each PLC was found within 19 days of the initial deployment. In 2020, Zhao *et al.* [22] evaluated five popular search engines, including Censys and Shodan, measuring their searching ability (how many devices each engine returns), raw data accuracy (ratio of valid data), response time (how long until a new device is indexed) and the scanning period (time difference between two contiguous scans). They found that Censys and Shodan had similar searching ability, and both are appropriate for users who wish to find newly exposed devices. The authors [22] recommended users who conduct research on recent data to use Shodan.

**Search engines' output processing.** Genge *et al.* [16] developed ShoVAT, a tool which processes the output of Shodan queries and compares it with the National Vulnerability Database [9] to detect vulnerable devices on the Internet. They found 3,922 known vulnerabilities across 1,501 services [16]. Ercolani *et al.* [15] used the data Shodan collects about devices to create visual models used to attempt to identify what the devices are.

## VII. Discussion and Concluding Remarks

Our empirical analysis provides evidence that both Censys and Shodan have the technical capability of reflecting updates from their banner grabs onto their search interfaces in a timely manner. Our analysis on HTTPS banner grabs shows that despite not capturing $\sim$90% of our daily-changed banners, remaining (captured) banners were made visible on Censys and Shodan search interfaces within a day of generating them. Shodan was slightly faster than Censys in grabbing new banners, and in making them visible through its search interface. Based on our experiment, both engines can benefit from an increased banner grabbing rate, which complement their efficiency in making new banners visible. Upon analyzing resource consumption, one can expect to receive $< 40$ banner grabs per month from each engine. Port scan sessions for open ports typically lasted less than a second, with Censys generally being faster than Shodan to close a session, especially for HTTP and HTTPS sessions. Banner grabs were slower, with many sessions lasting longer than 3 seconds. Finally, we noticed several differences in the operation of Censys and Shodan. For example, Shodan's scanners appear to be geographically scattered around the world, unlike Censys whose IP addresses are all in Michigan, USA; Censys appears more focused on HTTP/HTTPS than Shodan, and it appears to dedicate a specific port to certain scanners, and have other scanners probe many ports (*e.g.,* several hundred ports for a single scanner). Shodan's distribution of ports among scanners appears less skewed. Possibly a consequence of the previous point, the time of receiving Shodan's scanning traffic every day is slightly more consistent than Censys.

There are several avenues to expand upon the empirical work presented herein. For example, Censys/Shodan maybe scanning Amazon's AWS VMs different from non-Amazon devices. Additionally, our packet captures can be analyzed to understand how each engine addresses *diurnal effects*, *i.e.,* variations in scans to account for change in results based on the time of day [14]. Our HTTPS banner-grab freshness analysis can be extended to other services. In general, the data collection and analysis methodology used herein can also be applied to analyze non-TCP traffic from Censys and Shodan, like ICMP or UDP. Finally, we believe that more work is needed towards devising a systemic methodology for identifying traffic and source IP addresses of Internet-wide scanners that do not make this information publicly available.

## References


[1] Censys https://censys.io/.
[2] Shodan https://shodan.io/.
[3] The ZMap Project https://zmap.io/.
[4] Censys Contact Us https://censys.io/contact Accessed June 2019.
[5] SecurityTrails https://securitytrails.com/.
[6] Pcap-splitter 1.0.0 https://pypi.org/project/pcap-splitter.
[7] WhoisXMLAPI https://ip-geolocation.whoisxmlapi.com/.
[8] Cartopy 0.18.0 https://pypi.org/project/Cartopy/.
[9] National Vulnerability Database (NIST) https://nvd.nist.gov/.
[10] Internet Assigned Numbers Authority https://www.iana.org/.
[11] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the Ability of the Shodan Search Engine to Identify Internet-facing Industrial Control Devices," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114–123, 2014.
[12] Y. Chen, X. Lian, D. Yu, S. Lv, S. Hao, and Y. Ma, "Exploring Shodan From the Perspective of Industrial Control Systems," 2020.
[13] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-wide Scanning," in *ACM CCS*, 2015.

[14] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and its Security Applications," in *USENIX Security Symposium*, 2013, pp. 605–620.
[15] V. J. Ercolani, M. W. Patton, and H. Chen, "Shodan Visualized," in *IEEE Conference on Intelligence and Security Informatics (ISI)*, 2016.
[16] B. Genge and C. Enăchescu, "ShoVAT: Shodan-based Vulnerability Assessment Tool for Internet-facing Services," *Security and Communication Networks*, vol. 9, no. 15, pp. 2696–2714, 2016.
[17] H. Heo and S. Shin, "Who is Knocking on the Telnet Port: A Large-Scale Empirical Study of Network Scanning," in *ACM AsiaCCS*, 2018.
[18] A. Mirian, Z. Ma, D. Adrian, M. Tischer, T. Chuenchujit, T. Yardley, R. Berthier, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "An Internet-wide View of ICS Devices," in *IEEE PST*, 2016.
[19] J. Postel, "Transmission Control Protocol," Internet Requests for Comments, RFC Editor, Tech. Rep., September 1981, http://www.rfc-editor.org/rfc/rfc793.txt.
[20] S. Pritchard, "Shodan Founder John Matherly on IoT Security, Dual-Purpose Hacking Tools, and Information Overload," https://portswigger.net/daily-swig/shodan-founder-john-matherly-on-iot-security-dual-purpose-hacking-tools-and-information-overload Accessed Oct 2020.
[21] G. Wan, L. Izhikevich, D. Adrian, K. Yoshioka, R. Holz, C. Rossow, and Z. Durumeric, "On the Origin of Scanning: The Impact of Location on Internet-wide Scans," in *ACM IMC*, 2020.
[22] B. Zhao, S. Ji, W.-H. Lee, C. Lin, H. Weng, J. Wu, P. Zhou, L. Fang, and R. Beyah, "A Large-scale Empirical Study on the Vulnerability of Deployed IoT Devices," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2020 (to appear).

## APPENDIX

In Sec. IV, we analyze HTTP GET requests (banner grabs) from Censys and Shodan. Table III shows the user agents used by Shodan scanners, and Table IV shows the paths Shodan scanners requested. Censys scanners always used user agent "Mozilla/5.0 zgrab/0.x" and the path "/".

| Shodan User Agents | Count |
|---|---|
| No User Agent | 58 |
| Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36 | 20 |
| python-requests/2.23.0 | 2 |
| python-requests/2.10.0 | 9 |
| python-requests/2.11.1 | 1 |
| python-requests/2.19.1 | 5 |
| python-requests/2.22.0 | 2 |

TABLE III: Shodan user agents when requesting a resource from the HTTP service.

| Shodan Paths | Count |
|---|---|
| / | 20 |
| /robots.txt | 20 |
| /sitemap.xml | 19 |
| /.well-known/security.txt | 19 |
| /favicon.ico | 19 |

TABLE IV: Shodan HTTP resources requested.

In Table I, Sec. V-A, we listed the ports most commonly scanned by Censys and Shodan. In Table V (Censys) and Table VI (Shodan) we list the IANA [10] registered service for these ports, as well as which service a search on the respective engine returns when queried for the given port.

| Port | IANA Registry | Censys Search Result |
|---|---|---|
| 25 | SMTP | SMTP |
| 81 | - | - |
| 83 | MIT ML Device | - |
| 88 | Kerberos | - |
| 2082 | HTTP | - |
| 2083 | HTTP | - |
| 4567 | TRAM | |
| 5900 | Remote Frame Buffer | VNC |
| 5901 | - | VNC |
| 5902 | - | VNC |
| 8080 | http-alt | HTTP |
| 8081 | sunproxyadmin | - |
| 8088 | radan-http | - |
| 8090 | opsmessaging | - |
| 9200 | wap-wsp | Elasticsearch |
| 16993 | Intel(R) AMT SOAP/HTTPS | HTTPS |

TABLE V: List of Censys-scanned ports, and their IANA-associated services.

| Port | IANA Registry | Shodan Search Result |
|---|---|---|
| 25 | SMTP | SMTP |
| 53 | DNS | - |
| 81 | - | HTTP |
| 82 | xfer | XtremeRAT |
| 102 | iso-tsap | multiple |
| 444 | Simple Network Paging Protocol | SonicWALL |
| 1177 | DKMessenger Protocol | FileZilla Server |
| 1604 | icabrowser | OpenSSH |
| 1723 | pptp | OpenSSH |
| 2086 | GNUnet | HTTP |
| 2222 | EtherNet-IP-1 | HTTP |
| 3460 | EDM Manger | Unreal ircd |
| 4782 | - | OpenSSH |
| 5555 | Dual Stack MIPv6 NAT Traversal | OpenSSH |
| 5800 | - | RealVNC |
| 6666 | - | OpenSSH |
| 7443 | Oracle Application Server HTTPS | SonicWALL |
| 51235 | - | rippled |

TABLE VI: List of Shodan-scanned ports, and their IANA-associated services.