SEARCH ENGINES THAT SCAN FOR INTERNET-CONNECTED SERVICES: CLASSIFICATION AND EMPIRICAL STUDY

by

Christopher Bennett

A thesis submitted to the Faculty of Graduate and Post Doctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Computer Science

in

Network Security

Carleton University
Ottawa, Ontario

Abstract

In this thesis, we revisit outdated definitions of Surface Web and Deep Web and provide new definitions and apply them to Internet search engines. We argue that the scope of the term "Web" is too narrow when referring to information on the Internet. We offer, and define, new terms to better describe the state of the Internet: Surface Internet, Shallow Internet, and Deep Internet. We use these terms to describe: Responding Internet-Connected Entity (RICE), Search Engine for Responding Internet-Connected Entities (SERICE), Web search engines, and Internet search engines. We explain how popular Internet-wide scanning services — Shodan and Censys — are SERICEs that index RICEs. In empirical work, we analyze scans from Shodan and Censys and determine they use few resources and provide an up-to-date view of the Internet.

Acknowledgements

Throughout the writing of this thesis I have received a great deal of support and assistance. Firstly, I would like to express my gratitude and appreciation to my supervisors Dr. Paul van Oorschot and Dr. AbdelRahman Abdou for the long hours reviewing my thesis and mentoring me. I thank them for the lessons I have learned and will never forget.

I would like to thank the members of the Carleton Computer Security Lab and Carleton Internet Security Lab, especially Christopher Bellman, for their invaluable insight and discussions during my time at Carleton University.

I would also like to thank my love Stephanie Cameron for her relentless encouragement, support and putting up with me, especially during the COVID lockdown.

-Christopher Bennett

Table of Contents

Abstract . . . . . **ii**

Acknowledgements . . . . . **iii**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

Chapter 1 Introduction **1**

1.1 Search Engine (The Meaning of the Term) 1

1.2 Motivation 2

1.3 Contributions 2

1.4 Thesis Overview 3

1.5 Preliminary Background Definitions 3

Chapter 2 Characterizing Search Engines **9**

2.1 The Surface and Deep Internet 9
2.1.1 Surface Web, Deep Web, Dark Web and Darknets 9
2.1.2 Surface Internet 12
2.1.3 Deep Internet 14
2.1.4 Relationship Between the Shallow, Surface, Deep and Dark Internet 15

2.2 What is a Search Engine? 16
2.2.1 Search Engines 17

2.3 Taxonomy for Search Engines 19
2.3.1 Responding Internet-Connected Entity (RICE) 21
2.3.2 IP Address Crawler 22
2.3.3 Search Engine for Responding Internet-Connected Entities . . 23

2.4 Shodan 24
2.4.1 Shodan Web Interface 25

2.5 Censys 26

2.5.1 Censys IP Address Crawler (IPAC) 27
2.5.2 Censys Web Interface 27

2.6 Original Motivations for Shodan and Censys 28
2.6.1 Motivation for Shodan 28
2.6.2 Motivation for Censys 28

2.7 Shodan and Censys Use Cases and Related Work 28
2.7.1 Background 28
2.7.2 Shodan and Censys Use Cases 29
2.7.3 Discovering RICEs Infected with Known Malware 29
2.7.4 Identifying Vulnerable RICEs 31
2.7.5 Analyzing TLS Certificates 33
2.7.6 Analyzing Content Filters on the Internet 33

2.8 Conclusion 34

Chapter 3 Empirical Scanning Analysis of Shodan and Censys . . 35

3.1 Methodology 36
3.1.1 Setting Up Our Virtual Machines 36
3.1.2 Configuring Software on Our Virtual Machines 37
3.1.3 Identifying Foreign IP Addresses 38
3.1.4 Classifying Traffic 39

3.2 Results 40
3.2.1 Q1: How Fast do Shodan and Censys Update After a RICE Changes a Service's Banner? 40
3.2.2 Q2: How Much Traffic Can a RICE Expect to Receive from Shodan and Censys? 44
3.2.3 Q3: Which Services do Shodan and Censys Most Frequently Scan? 48
3.2.4 Q4: How Many Unique IP Addresses do Shodan and Censys Use for Scanning? 52
3.2.5 Q5: Do Shodan's and Censys' Scanners Located Geographically Near to a RICE Scan it More Frequently than Further Away Scanners? 54
3.2.6 Q6: What are the Scanning Patterns of Shodan and Censys? . 57
3.2.7 Q7: What Level of the Surface and Deep Internet do Shodan and Censys Index? 61

3.3 Limitations 63

3.4 Related Work 64

3.4.1 Identifying SERICEs on the Internet . . . 65
3.4.2 Studies that Evaluate or Measure the Functionality of SERICEs 66
3.4.3 Tools for Processing the Output of SERICEs . . . 67

3.5 Conclusion . . . 67

Chapter 4 Conclusion . . . 69

4.1 Contributions . . . 69

4.2 Open Problems With Internet-wide Scanning . . . 70
4.2.1 Server Name Indication Problem . . . 70
4.2.2 IPv6 Internet-wide Scanning . . . 71
4.2.3 Network Address Translation with IPv6 . . . 71
4.2.4 Dark Web Scanning . . . 72

4.3 Future Work . . . 72

Appendix A . . . 74

A.1 Transmission Control Protocol . . . 74

A.2 Shodan Domains . . . 77

A.3 Virtual Machine Setup Script . . . 85

Bibliography . . . 86

List of Tables

2.1	Search Engines Released Between 1990 and 2009	20
3.1	Virtual Machine Firewall Rules	36
3.2	Location and IP Address of Virtual Machines	37
3.3	Services Running on Our Virtual Machine	38
3.4	Five Most Common Open Port Scans	49
3.5	List of Ports Shodan Attempted to Scan	50
3.6	List of Ports Censys Attempted to Scan	51
3.7	Shodan Scanners Unique IP Addresses	53
3.8	Censys Scanners Unique IP Addresses	53
3.9	Distance of Scanners to Targets	55
3.10	Shodan User Agents	62
3.11	Shodan HTTP Resources Requested	62
A.1	List of Subdomains Registered for Shodan.io	78

List of Figures

1.1 Examples of TCP Port Scans 5
1.2 Example of a Banner from an HTTP HEAD Request 7

2.1 Surface, Shallow, Deep and Dark Internet 15
2.2 Example of Bing's and Google's SERPs 17
2.3 Relationship Between a SERICE and a Web Search Engine . . 19
2.4 Internet Connected Device with NAT 22
2.5 SERICE Depth Compared to Web Search Engine Depth . . . 24
2.6 Shodan Maps Example 26

3.1 Censys Scanner Locations on World Map 37
3.2 Time Differences Visualisation 41
3.3 Grab Span 42
3.4 Update Span 43
3.5 Refresh Span 43
3.6 Heatmap of SYN Scans 45
3.7 Heatmap of Banner Grabs 45
3.8 Heatmap of the Duration of SYN Scans 46
3.9 Heatmap of the Duration of Banner Grabs 46
3.10 Heatmap of the Standard Deviation of Time Between Scans . 47
3.11 Heatmap of the Standard Deviation of Time Between Banner Grabs 47
3.12 Number of TCP Sessions from a Shodan Scanner by Country the Scanner is Located in. 55
3.13 Shodan Scanner Locations on World Map 56
3.14 Censys Distinct Port Distributions of All TCP Sessions 58

3.15 Shodan Distinct Port Distributions of All TCP Sessions 58
3.16 Number of Distinct Ports Used for Open Port Scans 59
3.17 Number of Distinct Ports Used for Banner Grabbing 59
3.18 Distinct Port VM Distributions 60

A.1 The TCP Header 74
A.2 TCP Handshake Timing Diagram 77

Chapter 1

Introduction

The Internet contains an enormous amount of information, distributed across a vast number of devices, made accessible through many services running on those devices. In the early 90s, as the popularity of the Internet grew, so did the demand for a method for finding relevant, high-quality resources on the Internet. The invention of Web search engines fulfilled this requirement, and revolutionized how we find information. Search engines automatically collect information from the vast expanses of the Web, store the content they find and provide users with an interface to query these databases. Due to the size of the Web and the number of incoming queries, search engines must be able to gather information rapidly and keep their databases up-to-date. Search engines must handle a massive number of synchronous queries from users and produce relevant, quality results. Conventional search engines are capable of returning billions of relevant and high-quality results in a fraction of a second. However, conventional search engines are limited because they only search the Surface Web, a small subset of the Internet.

1.1 Search Engine (The Meaning of the Term)

The term *search engine* has been widely adopted as a software system that allows users to perform searches for Web content. *Search engines* crawl and index a massive database of publicly available websites. A user submits a query to the *search engine*, the *search engine* processes the semantics of the query, then returns a list of hopefully relevant links to the user.

Through this thesis, we expand the term *search engine* to include all services available on the Internet and not just websites. By services we mean a software program running on a computer that can communicate with a client software program (websites, file storage, web cameras, ssh servers, remote desktop hosts, etc.). This thesis

argues that the rise of Internet-wide scanning and Internet-wide scanning services has changed how we should think of *search engines*.

1.2 Motivation

The motivation of this thesis is to provide a taxonomy and classification for components of Internet-wide scanning that we feel are inadequately classified; and explore how they apply to popular Internet-wide scanning services. In our opinion the previous classifications of the "Surface Web" and "Deep Web" are both imprecise and outdated. We believe that our definitions provide a more useful taxonomy of the current landscape. Further more, we provide methodologies of how to evaluate Internet-wide scanning services and apply them to Shodan and Censys in our empirical analysis.

1.3 Contributions

The following are the contributions of this thesis:

1. We expand the definition of a *search engine* to include services that can crawl and index resources other than websites.

2. We define the following terms *IP Address Crawler*, *Responding Internet-Connected Entity (RICE)* and *Search Engine for Responding Internet-Connected Entities (SERICE)*.

3. We explore implementation issues and possible solutions related to Internet-wide scanning.

4. We carry out seven experiments to compare how Shodan and Censys conduct Internet-wide scanning.

In our work, we focus on Shodan and Censys as they are the two most popular search engines that conduct Internet-wide scanning. This thesis advances research in the area by classifying components of Internet-wide scanning and conducting an empirical scanning analysis of Shodan and Censys to determine how much resources are consumed and how fresh the results are.

1.4 Thesis Overview

This thesis is organized as follows. Chapter 2 presents our definition of what a search engine is, provides a definition for a *Responding Internet-Connected Entity (RICE)* and *Search Engine for Responding Internet-Connected Entities (SERICE)*, shows how these terms apply to Shodan and Censys and lists use cases of Shodan and Censys. Chapter 3 covers the results of our empirical scanning analysis of Shodan and Censys. Chapter 4 is the conclusion of the thesis, discussion of our results and mentions a few open problems in Internet-wide scanning.

1.5 Preliminary Background Definitions

We provide well known definitions of commonly used terms below for the convenience of the reader. Additional terms related to the Transmission Control Protocol (TCP) can be found in Appendix A.1.

Internet Protocol

The *Internet protocol (IP)* is the principal communications protocol used in the Internet [60]. The IP defines how blocks of data, called datagrams, are transferred from sources to destinations. Each host is identified by a fixed length address know as an IP address. Two widely used versions of the IP are: IPv4 and IPv6.

Internet Protocol Version 4 (IPv4) *IPv4* uses a 32-bit long address which provides a space of 2^{32} possible values [60]. This address space is too small to give each host on the Internet a unique IP address. This issue is known as *IPv4* address exhaustion.

Internet Protocol Version 6 (IPv6) *IPv6* uses a 128-bit long address which provides a space of 2^{128} possible values [32]. *IPv6* is intended to solve the issue of IPv4 address exhaustion.

Network Nodes A *network node*, *or node*, is any addressable device (physical or virtual) connected to a network.

Host A *host*, or *network host*, is a client or server (edge node) connected to a network. *Hosts* are assigned at least one network address. *Hosts* do not include network devices such as routers and switches. In our work, we categorize routers and switches as intermediate nodes. By these definitions, all *hosts* are nodes, but not all nodes are *hosts*.

Network Address A *network address* is a unique identifier for the connection point of a host or node on a network. Examples of network addresses include phone numbers, IP addresses and MAC addresses. This thesis only requires background on IP addresses.

Scanner A *scanner* is a host that attempts to collect data about one or more targets. *Scanners* generate an IP address, determine if a target exists at that IP address, enumerate the services hosted on the target, and collect application or protocol specific information about services running on a target.

Target A *target* is a host that a scanner collects data from. *Targets* can be physical machines or virtual machines. A *target* can be hidden behind a proxy or NAT that uses port forwarding to expose one or more of the *target's* services.

Port Scanning *Port scanning* is when a scanner attempts to determine if a target is running a service on a given port (either TCP or UDP). This thesis focuses on TCP scanning so we will only consider TCP port scanning. We consider any TCP stream that did not contain any packets during the ESTABLISHED TCP state to be a TCP port scan. This includes TCP streams that did not complete the TCP handshake. TCP Port scans are typically TCP streams comprised of a small number of packets with an empty data section. A few examples of TCP streams that would be considered TCP port scans can be found in Figure 1.1.

Port scanning allows the scanner to determine which ports the target device will accept network traffic on. Processes running on the device bind a port and listen for incoming connections. Processes representing standardized services typically bind to

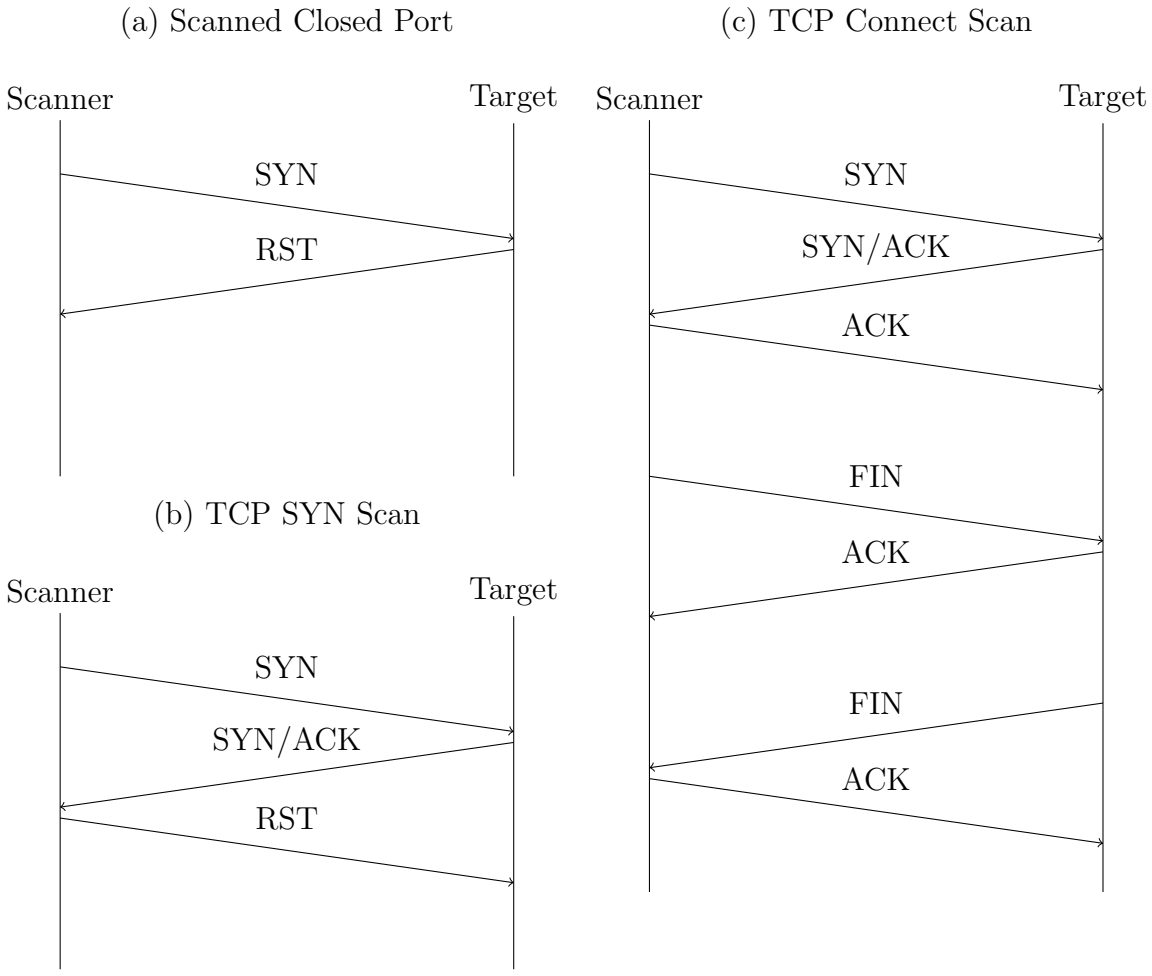


Figure 1.1: Examples of TCP Port Scans. (a) Attempted scan with closed port. (b) TCP SYN scan with open port. (c) TCP Connect Scan with open port.

a default port. For example HTTP services typically bind port 80, SSH services typically bind port 22 and HTTPS services typically bind port 443. However the Internet Assigned Number Authority (IANA) notes that traffic flowing to or from a registered port does not guarantee that the traffic corresponds to the assigned process [29]. Therefore the scanner must take into account the possibility that any application could bind to any available port. IANA classifies ports [0-1023] as System Ports, ports [1024-49151] as Users Ports and ports [49152-65535] as Dynamic or Private Ports [29]. Default System Ports are assigned and controlled by the IANA [64]. On some operating systems, the process must be running as a privileged user in order to

bind ports [0-1023]. This may increase the confidence a scanner would have that an open port between 0 and 1023 is running the well known service associated with that port.

Several methods of port scanning include *TCP NULL Scans*, *UDP Scans*, *TCP SYN Scans* and *TCP Connect Scans*. (See Figure 1.1.)

TCP NULL Scan *TCP NULL scan* has all control bit fields and the sequence number set to zero [27]. A scanner sends such a packet to the target device on the target port. If the port is closed, the device will respond with a RST. If the port is open, the device typically will not respond. The advantage of this form of TCP scanning is it may avoid being blocked by some firewalls. However, this scan does not allow for further querying of the device without creating a new TCP connection because the packet sent by a TCP NULL scan does not follow the TCP specification and will be discarded by the receiving host.

UDP Scan UDP scan is possible but is outside the scope of this thesis.

TCP SYN Scan *TCP SYN scan* is a fast method to determine if a port is open. The scanning machine creates a TCP packet with the SYN control bit set. The destination port set to the port number the scanner wishes to scan. The packet is sent to the target device and the scanner waits for a response. If the scanner receives a SYN/ACK from the server, the port is open and the scanner sends a reset to the server without completing the TCP 3 way handshake (Figure A.2). If no response is received, the scanner assumes the port is closed. This method does not guarantee the process bound to the port is the well known process. An example of TCP SYN scan can be found in Figure 1.1 (b).

TCP Connect Scanning *TCP connect scanning* relies on the underlying operating system to create a TCP session. A full TCP handshake is negotiated with the target device. Similar to the TCP SYN Scan, a TCP packet with the SYN control bit set is sent to the target device. If the target device responds with a TCP packet where the SYN and ACK control bits are set then the port is open. If no response

is received, or the target responds with a RST, then the scanner assumes the port is closed. Unlike the SYN scan, the scanner now sends another TCP packet with the ACK control bit set. Now that the TCP session is established the scanner can send network traffic to the process on the target device. An example of *TCP Connect Scan* can be found in Figure 1.1 (c).

Types of Port Scanning *Types of port scanning* are: vertical, horizontal and box scanning. A vertical scan is when a single IP address is scanned for multiple ports. A horizontal scan is when multiple IP addresses are scanned for a single port. Box scanning is a combination of both.

Banner Grab A *banner grab* is a technique used to collect a *banner* from a service, typically used after a port scan has identified that a service exists. A *banner* is information about a service, such as: version numbers of the service, keys or certificates used during setup of secure communication channels, the title of a webpage hosted by the service, the content of a webpage hosted by the service, the name of the software running the service, etc. The information obtained by a banner grab is typically parsed into what we will later call a *property* (Section 2.1.1 Page 9). An example of a *banner* returned from a *banner grab* (an HTTP HEAD request in this case) can be found in Figure 1.2.

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=ISO-8859-1
Date: Sun, 12 Nov 2020 10:24:32 GMT
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Expires: Sun, 13 Nov 2020 10:24:32 GMT
Cache-Control: private
```

Figure 1.2: Example of a banner from an HTTP HEAD request.

Internet-wide Scanning Internet-wide scanning (IWS) is when port scanning or banner grabs are attempted against as many hosts as possible on the Internet [36].

IWS of the entire IPv4 address space can be completed in as little as 45 minutes from user space on a single machine with a 1 Gbps connection [37]. At the time of writing, it is infeasible to conduct an IWS of all IPv6 addresses due to the 128-bit address space of IPv6 [59].

World Wide Web and HTTP *Hyper Text Transfer Protocol (HTTP)* [17] is an application-level protocol that allows the fetching of resources. It allows for the dissemination of images, files, CSS, JavasScript, Web APIs and HTML documents. *HTTP* has a secure extension known as *HTTP secure* (*HTTPS*), traffic sent over *HTTPS* is encrypted HTTP traffic.

The *World Wide Web (Web / WWW)* is an information system where Web resources, identified by Uniform Resource Locators (URLs), are made accessible via the Internet over the HTTP or HTTPS [44]. HTTP and HTTPS are the fundamental protocols in the WWW.

Chapter 2

Characterizing Search Engines

In this chapter, we first define three terms: *Surface*, *Shallow*, and *Deep Internet*. Then we revisit what a *search engine* is. We expand the definition of a *search engine* to include searching *Surface* and *Deep Internet*. Then we provide a description of what a *Search Engine for Responding Internet-Connected Entities* is, and how it relates to search engines. Finally, we provide two concrete examples of *Search Engine for Responding Internet-Connected Entities* — Shodan and Censys — and examine use cases for these two search engine services.

2.1 The Surface and Deep Internet

2.1.1 Surface Web, Deep Web, Dark Web and Darknets

The *Surface Web* has been described as the portion of the Web that a conventional search engine can index [26,40,67,74]. *Deep Web* is the rest of the Web that cannot be indexed. We believe that these definitions require revisiting for several reasons: there is no fixed definition of what a "conventional search engine" is and what a search engine can index changes dynamically. This does not allow for a precise definition of *Deep Web*. We propose new definitions below to replace these, defining Surface Internet in Section 2.1.2 and Deep Internet in Section 2.1.3.

The *Dark Web* is a subset of the *Deep Web* that has been intentionally hidden and cannot be accessed with a standard Web browser. The *Dark Web* comprises of the set of all *Darknets*. A *Darknet* is a network run on top of the Internet that requires special software, configurations or authorizations to access. *Darknets* often use a unique or customized communication protocol. Three popular *Dark Web* platforms are: Tor, Invisible Internet Project (I2P) and Freenet. We will use these definitions for *Dark Web* and *Darknet* for the remainder of the thesis.

In this section we introduce the terms *Shallow Internet*, *Surface Internet* and *Deep Internet*. We emphasize that we purposely chose the term "Internet" instead of "Web", because we will be applying these terms to services other than the Web. Our goal is to provide concrete definitions for these terms that do not rely on what a search engine can find. Figure 2.1 (Page 15) shows the relationship between *Shallow*, *Surface* and *Deep Internet*. First, we define three terms to help us describe the Surface, Shallow, and *Deep Internet*: property, *resource* and *Uniform Resource Locator*. (Note: rather than proposing new definitions of resource and URL, we are simply providing standard definitions as a convenience to make our discussion self-contained.) In this section a "user" means any entity that wishes to obtain a resource (can be a human or a robot).

Property

In this thesis, we define a *property* as name-value pair. The name must be a string (surrounded by quotes). The value can be: itself a *property*, string (surrounded by quotes), integral type, floating point, character, byte or an array of any of the previous. Our recursive definition allows for a *property* to be the value of a *property* (for example, consider the *property*: {"HTTP" : {"version" : 1.1}}). We write *properties* in the format {name : value}. We write arrays, where v is a value, in a comma separated format: $[v_0, v_1, \ldots, v_n]$. Properties are typically created (or extracted) by parsing a banner. We define three distinct types of *property* below: *location properties*, *encryption properties*, and *attribute properties*.

Example of *properties*:

- {"IP address" : 123.123.0.111}
- {"page_name" : "Welcome to my site"}
- {"search_results" : [{"service" : "HTTP"}, {"port": 8080}]}
- {"ssl.cert.header" : [{"version" : 3}, {"serial number" : 123456}]}

Location Properties We define *location properties* to be properties that contain information about how to connect to a target service. This includes IP addresses,

URLs, domain names for SNI (described in Section 4.2 on Page 70) and transport level context such as TCP or UDP port numbers. Some examples of a location are: a website located at URL "www.mysite.com:1234" and an SSH server reached with "username@123.123.123.111".

Encryption Properties We define *encryption properties* to be properties that contain information about the encryption being used, if any, to decrypt traffic received from the service and encrypt traffic sent to the service. This includes all information about which encryption algorithm is being used, certificate serial numbers, signature algorithms, public keys, version numbers, etc. Not all Internet-connected devices have *encryption properties*. An example of the "Certificate" *encryption property* of an HTTPS website can be found below.

```
{"Certificate": [
    {"Data":
        [{"Version": 3},
        {"Serial Number":
            "b1:ac:d2:f1:2a:34:3b:6f:c9:98:55:25:3d:dd:fa:03:07:f1"}]},
    {"Signature Algorithm": "sha256WithRSAEncryption"},
    {"Issuer": ["C=US", "O=Let's Encrypt", "CN=Let's Encrypt Authority X3"]},
    {"Validity" : [
            {"Not Before": "Nov 14 13:56:53 2020 GMT"},
            {"Not After" : "Jan 26 13:46:23 2021 GMT"}]},
    {"Subject": "CN=abc.mysite.ca"},
    {"Subject Public Key Info": [
        {"Public Key Algorithm": "rsaEncryption"}
        {"Public-Key": "(2048 bit)"}]}
}
```

Attribute Properties We define *attribute properties* to be properties that contain information about a device or services running on the device that are not location properties or encryption properties. Examples of *attribute properties* are an HTTP server running "Apache httpd"

Resource

As defined by RFC3986 [16], a *resource* is a general term to refer to an electronic document, an image, a source of information, a service or a collection of other *resources*. A *resource* can be represented as a collection of properties. For example, a webpage is a *resource* and can be represented by the property {"html.body" : "<html><body>Hello World!</body></html>"}.

Uniform Resource Locator (URL)

As defined by RFC3986 [16], a URL identifies a resource and contains the information required to locate the resource on the Internet.

Some examples of URLs:

https://www.mysite.com

ftp://ftp.mysite.com

https://admin:hunter2@admin.mysite.com/secret.php

https://mysite.com/index.php?userId=1234&SessionKey=A15D2F1

A URL consists of the following components [16]:

```
URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part   = "//" authority path-abempty
              / path-absolute
              / path-rootless
              / path-empty
```

2.1.2 Surface Internet

We define the *Surface Internet* as the portion of the Internet from which resources can be retrieved, without restrictions, with a URL, and including two aspects: properties and resources.

Shallow Internet

We define the *Shallow Internet* as all resources available on the Surface Internet. The Shallow Internet is a subset of the Surface Internet because it excludes properties. We partition the *Shallow Internet* into three distinct categories: *Default Resources*, *Discoverable Resources* and *Obscure Resources*.

Default Resource

We define a *Default Resource* to be a resource that can be retrieved with a URL that contains an empty path. As an example, we consider both of the following to be *Default Resources*: the default homepage of a website and the banner of an SSH server (the banner can be configured to display before a user logs in). Both are presented to a user who need know no more than the IP address or URL when they attempt to visit. Not all services have a *Default Resource*.

Discoverable Resource

We define a *Discoverable Resource* to be a resource that can be retrieved with a URL that does not have an empty path. As an example, consider a website that, on the homepage, has a link to an about page and contacts page. The about and contacts pages are *Discoverable Resources*. A user would browse to the homepage, see the links, and be able to click on them to access the resources.

Obscure Resource

We define an *Obscure Resource* to be a resource that is difficult for a user to find naturally (no link provided on the website), however access to the resource is not otherwise restricted. For example, consider a webpage with a difficult to guess URL such as any of the following examples:

- A hard to guess URL: “www.mysite.com/aXJvbm1hbg.html”
- A GET request: “www.mysite.com/tool.php?id=098116119” (This assumes a different value for the id variable causes a different resource to be served.)

- URL containing username and password: ssh://admin:hunter2@mysite.com

2.1.3 Deep Internet

The *Deep Internet* is the portion of the Internet from which resources cannot be retrieved with a URL alone but also require extra information. (We will explain "extra information" in what follows.) We breakdown the Deep Internet into four categories: *Secluded Resources*, *Obstructed Resources*, *Protected Resources* and Darknets.

Secluded Resource

We define a *Secluded Resource* to be a resource that is not retrievable with only a URL because it requires extra information that cannot be passed in a URL. The extra information must not be a password. An example of a *Secluded Resource* is a website form that asks a user to input which country they are from. The value of the country is passed in a POST request and the site responds with the page associated with that country. Obscure resources and *Secluded Resources* are similar, however obscure resources are reachable with only a URL while *Secluded Resources* are not.

Obstructed Resource

We define an Obstructed Resource to be a resource where access is restricted, by the service hosting the resource, based on attributes of the user. The hosting service can accomplish this by explicitly blocking a user with a blacklist or implicitly blocking a user because they are not on a whitelist. For example, consider a service that blocks users based on: IP address, country they appear to be from (IP address based geographical location), the operating system or software the user is using (HTTP(S) identifies this with a "user agent" field in the request), etc.

Protected Resource

We define a *Protected Resource* to be a resource that requires a password, a session key or other means of proving a user's authorization to access. Examples of this are: a website that requires a user to login to an account, an SSH service that requires a user

to identify a pre-registered public key (that provides some degree of authorization) and prove they have the corresponding private key, or an FTP server that requires the user to login.

Darknets

Darknets (defined in Section 2.1.1) are outside the scope of this thesis, but Darknets could be broken down further within our model. Tor is an anonymity network which also features a darknet [9].

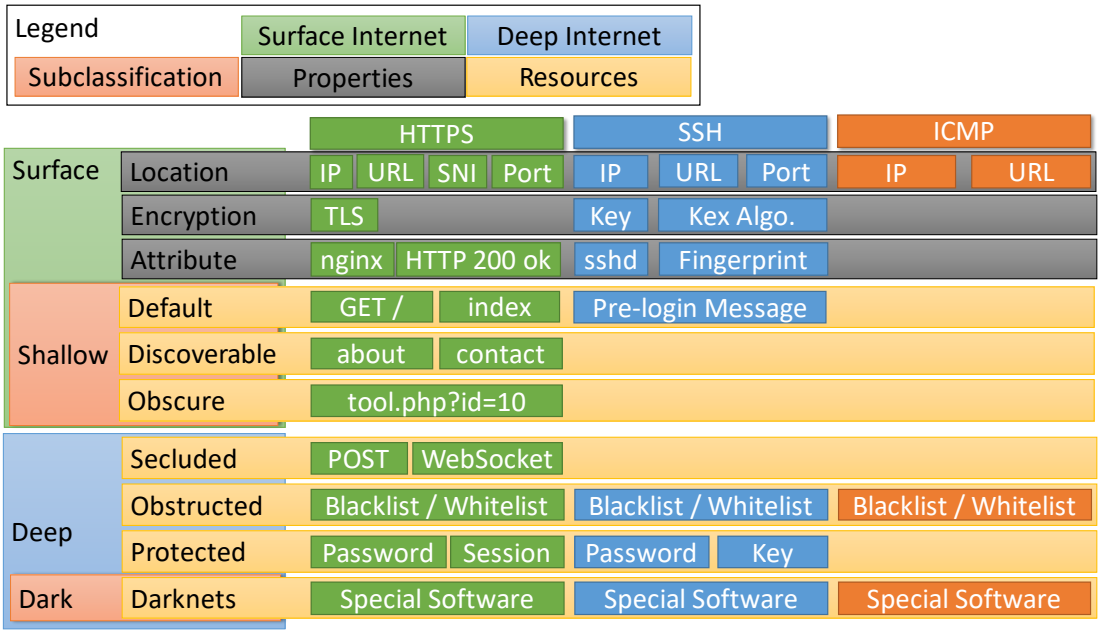


Figure 2.1: Relationship between the Shallow, Surface, Deep and Dark Internet. The relationship is illustrated for three protocols: HTTPS, SSH and ICMP. Properties and resources are listed in order from shallowest to deepest. In our model the Deep Internet does not contain properties. The lower a property or resource is on the chart the harder it is to find from an Internet-wide scanning perspective.

2.1.4 Relationship Between the Shallow, Surface, Deep and Dark Internet

This subsection serves as a walk-through for Figure 2.1. The rows of the figure contain labels of the properties and subclassifications of Surface Internet and Deep Internet

that we defined in this section. The columns have three services: HTTPS, SSH and ICMP. The intersection of a row and a column contains one or more labels that represent either a property(black background) or a resource(yellow background) and shows which level it belongs to. These properties and resources are not an exhaustive list of all possible properties or resources that fall under our classification. They serve as examples to help the reader understand our classifications.

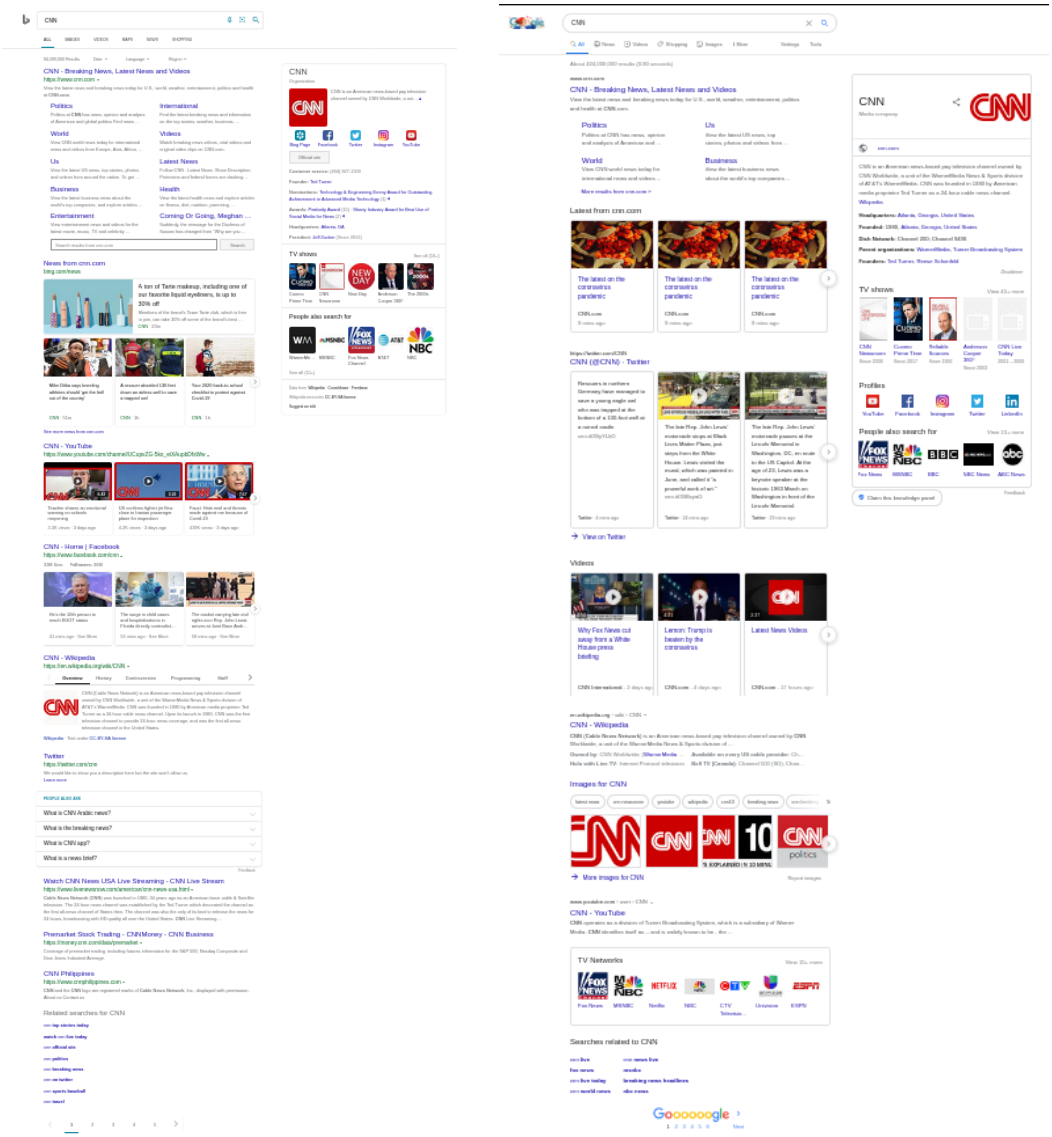
2.2 What is a Search Engine?

From a review of the literature [23, 30, 34, 58, 68], we suggest that the key features that define a *search engine* are: *Crawling*, *Indexing*, *Ranking* and *Querying*. Our classification will be discussed further in this section after additional definitions. We list a few services historically labeled as *search engines* in Table 2.1. We note that all conventional *search engines* support crawling, indexing, ranking and querying. We also point out that the majority of the *search engines* in this table target the Web. We will introduce our own definitions for *crawling*, *indexing*, *ranking* and *querying*. In this section, we argue that *search engines* are not just for the Web.

Many people associate the term search engine exclusively with popular services such as Google and Bing *search engines*. In this thesis, we distinguish between a "search engine" and a "Web search engine". We propose that a Web *search engine* is a *search engine* but a *search engine* is not always a Web *search engine*. We classify Google and Bing as Web *search engines* as they primarily search the Web. (By Web we mean HTTP and HTTPS services running on port 80 and 443 respectively.)

Search Engine Results Page

Search Engine Results Page (SERP) is the webpage that is sent to a user after they have submitted a query to a search engine. Examples of two *SERPs* for the query "CNN" on Google and Bing Web search engines can be found in Figure 2.2.



(a) Bing SERP (b) Google SERP

Figure 2.2: Example of a Bing SERP (a) and Google SERP (b) for the query "CNN" (images taken July 27th 2020). The small font text content is not intended for reading.

2.2.1 Search Engines

In this subsection we provide definitions for crawling, indexing, ranking, querying and search engine, based on a review of the literature [23, 30, 34, 58, 68].

Crawling and Crawlers

A *crawler* is a generic term for software that systematically discovers the location of new online resources, collects information from online resources, and revisit previously known locations to check if the resources have been updated. We explain the systematic methods for Web crawlers and Internet crawlers in Section 2.3 (Page 19). One systematic method a crawler may implement is *enumeration*. *Enumeration* is the complete listing of elements of a search space, in some fixed (i.e., specific) ordering. This is effective when an enumeration of the address space is possible in a reasonable amount of time. For example the enumeration of all IPv4 addresses is possible in under 45 minutes [37]. At the time of writing this thesis, the search space of IPv6 is too large for enumeration to be practical. Our definition allows for crawlers that use methods other than enumeration to be considered crawlers.

Indexing

Indexing is the process of analyzing and organising resources in order to allow for efficient and fast retrieval when a query is submitted to the database. *Indexing* dramatically speeds up the process of locating a resource in a database and retrieving it. It is common for Web search engines to *index* a webpage based on content that is more likely to be relevant to a user's query such as the title of the page, headings, bold words and meta tags.

Ranking

Ranking is the algorithm by which a search engine orders the results on its SERP, from most relevant to least.

Querying

Querying allows a user to use a text-based query to search for relevant, quality results. The user's query may be a natural language statement (eg. "What is the weather like today?") or follow a fixed structure (eg. "weather.get(date.today)").

Search Engine

A *search engine* is software which, given a user's query, attempts to retrieve the location of relevant resources and resource meta-data from its database and return them to the user. Queries to a *search engine* do not uniquely identify a single resource. Instead, the *search engine* returns multiple resources and displays them in an order based on the ranking. The goal of a *search engine* is to allow users to find the URL of a resource and a summary of the information that is relevant to the user's query. Some *search engines* support boolean operations such as AND, OR and NOT to allow users to create queries that will return the most relevant resources to them. By our definition, a *search engine* must support crawling, indexing, ranking and querying. Query results are displayed on a search engine results page (SERP).

2.3 Taxonomy for Search Engines

In this section we define the terms *Web Search Engine*, *Web Crawling* and *Responding Internet-Connected Entity*. A visual relationship of these terms can be found in Figure 2.3.

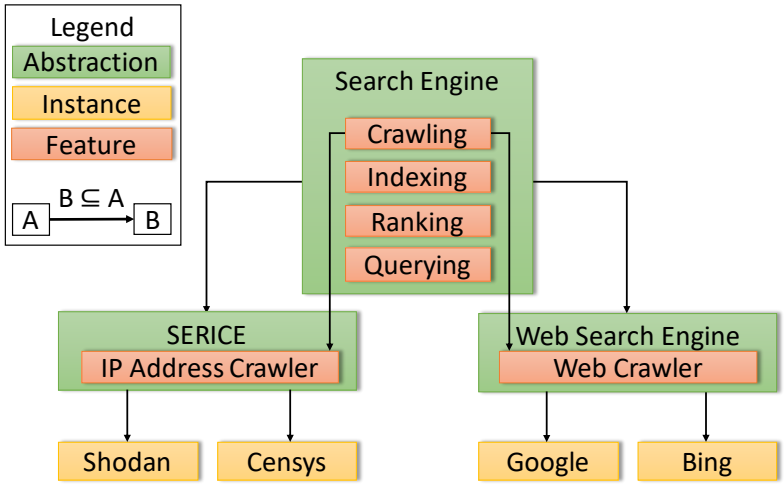


Figure 2.3: Relationship Between a SERICE and a Web Search Engine

Search Engine	Release Year	Searches	Crawling	Indexing	Ranking	Querying
Archie	1990	FTP			✓	
Veronica	1991	Gopher		✓	✓	
Jughead	1991	Gopher		✓	✓	
W3Catalog	1993	Web			✓	✓
WWW Wanderer	1993	Web	✓	✓	✓	
AliWeb	1993	Web			✓	✓
Jump Station	1993	Web	✓	✓	✓	✓
Web Crawler	1994	Web	✓	✓	✓	✓
Lycos	1994	Web		✓	✓	✓
Infoseek	1994	Web	✓	✓	✓	✓
AltaVisa	1995	Web	✓	✓	✓	✓
Inktomi	1996	Web	✓	✓	✓	✓
Dogpile	1996	Meta			✓	✓
HotBot	1996	Web	✓	✓	✓	✓
Ask Jeeves	1996	Web	✓	✓	✓	✓
Northern Light	1996	Web	✓	✓	✓	✓
Google	1998	Web	✓	✓	✓	✓
Teoma	2000	Web	✓	✓	✓	✓
Vivisimo	2000	Web	✓	✓	✓	✓
Exalead	2000	Web	✓	✓	✓	✓
Yahoo! Search	2004	Web	✓	✓	✓	✓
MSN Search	2005	Web	✓	✓	✓	✓
Good Search	2005	Web	✓	✓	✓	✓
Bing	2009	Web	✓	✓	✓	✓

Table 2.1: List of search engines released between 1990 and 2009. Compiled from [68].

Web Search Engine We define *Web search engine* to be a search engine that is primarily used to find resources on the Web using Web crawling (defined below). The majority of existing search engines primarily target the Web. For this reason, we explicitly call them *Web search engines*. We note that many *Web search engines* support non-Web searches such as maps, airlines, images, videos or finances; however, the main functionality is to search for resources on the Web. Two examples of popular Web search engines are: Google and Bing. The distinguishing feature of a *Web search*

engine is that it uses a Web crawler.

Web Crawling We define *Web crawling* as the process of collecting online resource from the Web. An entity that performs *Web crawling* is a Web crawler (sometimes known as Web spider, Web robot, Web wanderer or automatic indexer). *Web crawlers* find webpages that are uniquely identified by a URL. Hyperlinks from one webpage to another are a directed connection between the webpage containing the link to the webpage the link refers to. Web crawlers start with one or more seed URLs [23], download the webpage the URL points to, parses the HTML to find links on the page, then follow the new links to new pages and resources. Websites may use the "Standard for Robots Exclusion" [47] by including a "robots.txt" file to the root directory of the website. The robots.txt document specifies the rules that Web crawlers are requested to follow when accessing their website. Resources gathered from Web crawlers are indexed in order to be useful to the Web search engine.

2.3.1 Responding Internet-Connected Entity (RICE)

We define a *Responding Internet-Connected Entity* (*RICE*) to be any host that is reachable by a public IP address and will respond to at least one packet sent to it. By our definition, if a host is TCP port scanned on a closed port and responds with a RST response, then it is a *RICE*. There is no requirement for a *RICE* to use TCP or UDP. For example, a host that responds to an ICMP echo request is a *RICE*. Intermediate network nodes such as routers, cable modems or other networking devices, can be *RICEs*. However, a layer two switch that is not addressable by IP address is not a *RICE*. A *RICE* is uniquely identifiable (thus there is a one-to-many mapping between a *RICE*, defined by an IP address, and services or devices) by its public IP address. If the public IP address is reassigned to a new host, the *RICE* now refers to that new host. It is important to note that several devices may be reachable through network address translation (NAT) or a proxy. Since all of these devices are addressable via the same public IP address (even though they have separate ports), they would be considered one *RICE* as a single device running these services would appear identical from the perspective of a SERICE. This is shown in Figure 2.4.

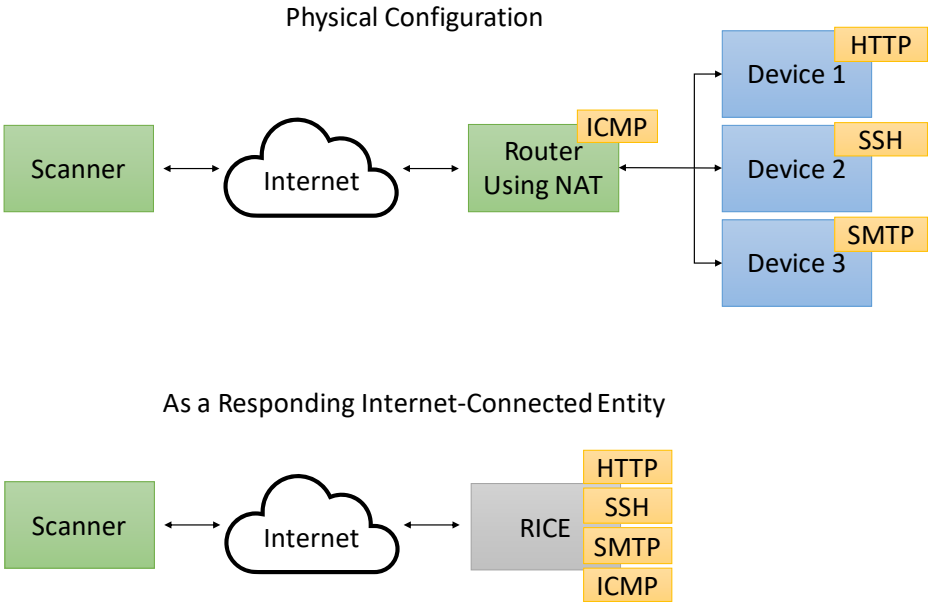


Figure 2.4: Demonstrates the difference between how a NAT is actually configured, and how it would appear as a RICE from a scanning perspective. Green and blue rectangles represent devices. Green rectangles are devices with public IP Addresses. Blue rectangles are devices with private IP addresses. Yellow rectangles indicate a service visible to a port scan.

2.3.2 IP Address Crawler

We define an *IP Address Crawler (IPAC)* to be a type of crawler (defined in Section 2.2.1) that conducts Internet-wide scanning (Section 1.5 Page 7). An *IPAC* generates a candidate IPv4 address (typically by enumeration like ZMap [37]) or IPv6 (this is an open problem in IWS due to the enormous search space of IPv6) or both.

By our definition, *IPACs* focus on collecting properties from the Surface Internet. *IPACs* index RICEs based on the responses received from the *IPACs* scans. This is commonly TCP and UDP port scanning, but can also be ICMP packets or any other protocol that would respond. *IPAC* port scanning has two main phases: port scanning and banner grabbing. While it is possible to conduct scanning on link layer protocols, we do not consider this to be Internet-wide scanning and therefore such scanning is not covered by this thesis.

We intentionally use the term "crawler" to describe an *IPAC* to allow for this definition to fit future addressing schemes. We considered using the term "IP Address Enumerator" but the in-progress migration to IPv6 addressing will make enumeration infeasible.

The two main differences between a Web crawler and an IPAC are: a Web crawler only targets HTTP and HTTPS services while an IPAC targets many services, and a Web crawler attempts to index resources while an IPAC focuses on indexing properties.

2.3.3 Search Engine for Responding Internet-Connected Entities

We define a *Search Engine for Responding Internet-Connected Entities (SERICE)* to be a search engine that uses an IPAC to crawl the IP address space discovering RICEs and then indexing properties (SERICEs focus on properties) and resources found. *SERICE* primarily targets Default Resources and Discoverable Resources on the Surface Internet. Figure 2.5 shows the differences between the depth of Internet that *SERICEs* and Web search engines target. Two popular *SERICEs* are Shodan and Censys, they are the main focus of the remainder of this thesis; numerous other *SERICEs* exists, e.g., Zoomeye [13].

In 2008, Madhavan et al. [54] describe a system for *surfacing* (taking Deep Web resources and indexing it) Deep Web content. They focused on automatically filling in HTML forms and submitting the data to the server. They note that when an HTML form is submitted, the values of the form get submitted as one of two methods: GET or POST. When the form is submitted as a GET request the parameters are appended to the URL.

The example they give is:

```
http://jobs.com/find?src=hp&kw=chef&st=Any&sort=salary&s=go
```

By our definition, this is an example of Google search engine indexing an Obscure Resource because the URL contains fields that are difficult to guess.

When the form is submitted as a POST request, the resulting URL does not uniquely identify the returned resources and hence cannot be indexed. The authors [54] have shown they are able to discover(by our definition) Secluded Resources

but they are unable to index them because the search engine identifies web pages based on their URLs (they note that URLs from GET requests are unique while URLs from POST requests are not).

In Section 3.2.7 (Page 61), we show results in our empirical study that suggests that Shodan (by our definitions a SERICE) is indexing Default Resources as well as Discoverable Resources.

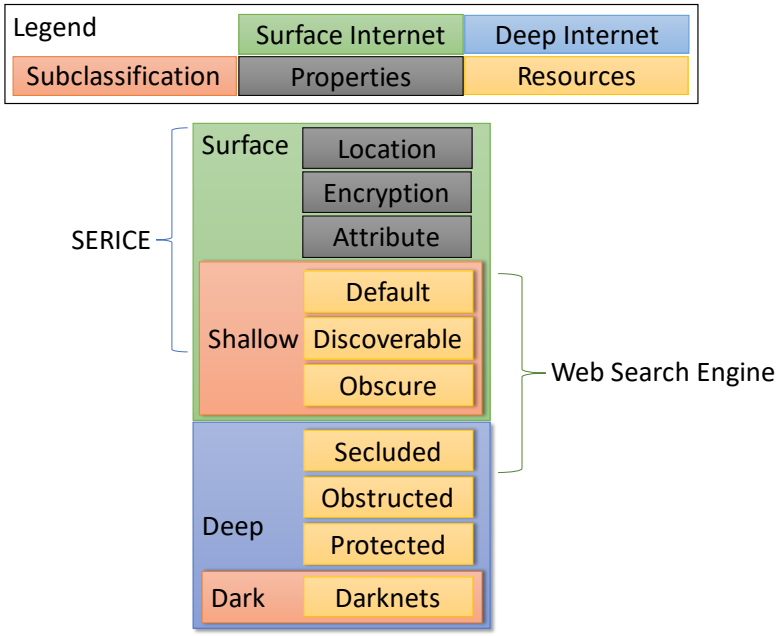


Figure 2.5: Levels of Surface and Deep Internet each search engine is able to discover.

2.4 Shodan

Shodan, which by our definition is a SERICE, was created in 2009 by John Matherly, to provide a "complete snapshot" of the Internet [10]. It accomplishes this by conducting Internet-wide scans, searching for, what we have defined to be, RICEs. Shodan attempts to identify any services the RICE is running and collects properties and resources from these services. Examples of properties Shodan collects are: the IP address, port numbers, headers and banners. Shodan accepts search queries through two main interfaces: Shodan's Web interface and Shodan's API (a non-interactive interface). Shodan is available at `https://www.shodan.io` and the API documentation

is available at `https://developer.shodan.io/`, as of December 2020.

2.4.1 Shodan Web Interface

Searching Using Shodan

Shodan's website's index page contains a search bar that allows a user to browse data via a Web search interface. The user enters a query and Shodan responds with a SERP. The SERP contains a list of IP addresses that correspond to RICEs that matched the user's query. The SERP provides information about the RICEs such as: location, Internet service provider, last time it was scanned by Shodan and global statistics (most popular countries (including a heat map), services, organizations, operating systems and products). Any IP address, on the SERP, can be clicked on and the user will be taken to a detailed page about that RICE. The page includes information such as country, organization, Internet service provider, last time the entry was updated, list of all open ports, a list of services running on the open ports, a list of Web technologies the device is running and a list of vulnerabilities that may affect the device.

Shodan Maps

Shodan Maps [55] is another Web search interface but, instead of returning a list of properties of a RICE, returns a visual representation of where each RICE is, physically on a map. This feature allows a user to use the same queries and syntax as the main search bar on the Shodan homepage. Figure 2.6 shows an example of a Shodan query. The query string is "city:ottawa country:CA SSH", meaning that the user is seeking information about SSH servers located in Ottawa, Canada. In Figure 2.6, each red dot represents one or more RICEs. Shodan Maps is able to display up to 1000 RICEs at a time [55]. We can see Shodan returned a total of 8,854 RICEs for our query, of these, 8,280 were running SSH. This is possible because our query has the string "SSH" which means, any banner containing the string "SSH" would match our query. For example, a RICE running an HTTP service where the index.html page contained the string "SSH" but has no SSH service running could match this query. We can see the most popular services that matched our query, in order, are: SSH, port 2222

(SSH alternate port), HTTPS (some of these RICEs are running SSH on port 443), port 4118 (WatchGuard SSH) and port 830 (OpenSSH). The "Top Organizations" are the organizations that own the netblock (a range of consecutive IP addresses) the RICE belongs to.

The Shodan API allows users to programmatically submit queries to Shodan. Everything that can be accomplished on the Shodan Website is possible using the API [55].

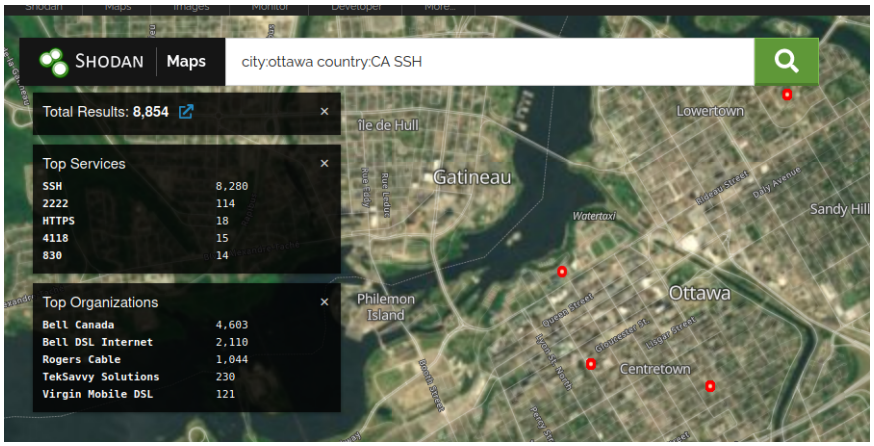


Figure 2.6: Shodan's Map SERP of the SSH RICEs located in Ottawa, Canada. Accessed Nov 29, 2020. As noted inline, one red dot may represent many RICEs.

2.5 Censys

Censys, which by our definition is a SERICE, is a platform designed to help information security practitioners discover, monitor and analyze devices on the Internet. Censys was founded by security researchers David Corcoran, David Adrian, Zakir Durumeric and J.Alex Halderman [2]. Originally, the Censys team's goal was to provide researchers with a tool to allow measurement to determine whether Internet security was improving. In order to track security vulnerabilities on RICEs, they created the "ZMap Project" [12] (not to be confused with the ZMap network scanner, which is a part of the ZMap Project). ZMap Project is a collection of open source tools that enable large-scale studies of services on the Internet [37]. They developed Censys using tools from the ZMap Project.

2.5.1 Censys IP Address Crawler (IPAC)

Censys' IPAC is comprised of two pieces of software from the ZMap Project: ZMap and ZGrab.

ZMap is an open source, modular network scanner specifically designed to perform Internet-wide scans [37]. It is capable of scanning the entire IPv4 address space, for a single port, in under 45 minutes on a gigabit Ethernet connection. ZMap scans IPv4 addresses according to a random permutation of the IPv4 address space to reduce the likelihood of overloading a destination network, which means more consistent results [37]. ZMap is used to determine if a port is open on a target. The next step is to collect properties from the services running on the identified ports; to do this, they use ZGrab.

ZGrab is a stateful application-layer scanner [12]. This is Censys' banner grabbing tool. It is responsible for taking the output from ZMap and conducting banner grabs on the open ports. We explain banner grabs in Section 1.5 on Page 7.

2.5.2 Censys Web Interface

Censys Web interface allows a user to browse data by submitting search queries on their site. Users must first select one of three datasets to search: IPv4 hosts, websites or certificates [3]. Similar to Shodan's Web interface, the user enters a query and Censys responds with a SERP. The SERP contains a list of IP addresses that correspond to RICEs that matched the user's query. Each IP address is listed next to a summary of some of the data collected about the corresponding RICE such as: autonomous system name, location, protocols, homepage title, global statistics or names on certificates. Similar to Shodan, Censys also provides a map SERP displaying the location of RICEs. Censys provides an API that accepts any query programically that can be performed through Censys' Web interface. The following query searches for RICEs running an HTTP service on port 80, where the title field contains the word admin.

```
80.http.get.title: admin
```

2.6 Original Motivations for Shodan and Censys

While the motivations for these SERICEs are not a technical observation, they help us understand why these tools exist and provides insights about intended use cases.

2.6.1 Motivation for Shodan

In an interview with The Daily Swig [62] John Matherly states "The goal was to offer a Netcraft on steroids: provide real-time market intelligence about everything that's on the Internet." Matherly said Shodan was not originally designed for security but instead for market intelligence [62]. When Shodan originally launched, Matherly advertised it as "Shodan, the search engine for hackers".

2.6.2 Motivation for Censys

Censys' creators, Durumeric et al. [35], state that Internet-wide scanning is a labor-intensive process and that answering simple questions about the state of the Internet can take weeks of effort. They comment on how researchers wishing to conduct Internet-wide scanning must consult with legal and networking resources before they can scan. Since many researchers and institutions lack access to these resources they created Censys to democratize Internet-wide scanning by providing an approximate real-time view of the Internet as an online, publicly available, service. The creators make it clear that Censys does not perform login attempts, attempt to exploit or access non-public resources.

2.7 Shodan and Censys Use Cases and Related Work

In this section, we first review two background terms and then cover use cases of Shodan and Censys.

2.7.1 Background

Cryptojacking *Cryptojacking* is the unauthorized use of cryptomining software to mine cryptocurrency [24]. An example of a *cryptojacking* attack is compromising

a website and injecting JavaScript code into a site that causes a victim's browser to mine cryptocurrency while browsing the site.

Remote Access Tool A *remote access tool (RAT)* is software used to remotely access and control a computer. *RATs* have both legitimate and malicious use cases. A systems admin could use a RAT to access a computer for configuration or to install new software. Cyber criminals can use RATs to control and monitor a victim's computer.

2.7.2 Shodan and Censys Use Cases

We distinguish the following categories for Shodan and Censys usage:

- Discovering RICEs infected with known malware
- Identifying vulnerable RICEs
- Measuring TLS certificates (i.e., collecting related current and historical metadata)
- Analyzing content filtering (resources are restricted by a third-party) on the Internet

We have grouped papers, from the literature, into these categories, and discuss them in the subsections immediately below. For the purposes of this section, we consider the use of ZMap to be a use of Censys. Since we defined the term RICE after these papers where written, we will use the term the authors used to describe entities connected to the Internet.

2.7.3 Discovering RICEs Infected with Known Malware

Shodan

In 2018, Rezaeirad et al. [65] reported on attackers and victims of two popular remote access tools: njRat and DarkComet. They note that Shodan supports active probing and banner identification for both. They found that the RATs were used primarily by operators located in the same country as the victims.

Censys

In 2017, Antonakakis et al. [15] provided a retrospective analysis of the Mirai botnet's growth over seven months. They used Censys to analyze how the Mirai botnet emerged, determine what classes of devices were infected, which manufacturers made the devices, and analyze how variants of Mirai evolved.

Shodan and Censys

In 2017, Farinholt et al. [39] studied the use of a popular RAT, "DarkComet". When an infected host establishes a TCP connection with a DarkComet controller, the controller sends a banner that identifies itself. Farinholt et al. use this to determine if a host is a DarkComet controller. They utilized ZMap to scan TCP port 1604 (DarkComets default controller port) twice per day to identify IPv4 addresses that are potentially hosting a DarkComet controller. They also parsed Shodan's daily response logs looking for the DarkComet banner on port 1604. They compiled the results from ZMap and Shodan to create a list of suspected DarkComet controllers. Every 30 minutes, they attempted a banner grab on each suspected DarkComet controller. In total they identified 9877 total DarkComet controllers, ZMap identified 56% of the devices and Shodan's logs identified 8% of the devices.

In 2019, Bijmans et al. [18] report on a new attack vector for cryptojacking (defined in Section 2.7.1 on Page 28). A firmware vulnerability in MikroTik routers allows cyber criminals to embed cryptomining code in any outgoing Web connection. They used both Shodan and Censys to discover infected routers. Using Censys they searched for TCP port 80 and 8080 (HTTP and HTTP alternative ports), and determined if the device was infected by checking if the HTTP header contained cryptomining code. Censys found a total of 1,452,550 unique IP addresses of infected routers (approximately 70% of all deployed MikroTik routers). Given the results from the Censys search, they queried the compromised IP addresses with Shodan's API to search historical records to determine the timestamp of when Shodan first encountered the infected routers.

2.7.4 Identifying Vulnerable RICEs

Shodan

In 2017, Böhme et al. [21] used Shodan to identify 250k devices vulnerable to Heartbleed (as of April 2016).

In 2019, Wang et al. [73] were able to identify vulnerable Internet of Things devices without having access to the device's hardware or firmware. They identified Internet of Things companion apps, downloaded them, and fed them into their "App Analysis Engine". The App Analysis Engine is able to identify possible vulnerabilities on a device that the app was designed to control. Using Shodan, they discovered 58,456 devices that are potentially vulnerable to at least one of: CVE-2017-8221, CVE-2017-8222, CVE-2017-8223, CVE-2017-8224 or CVE-2017-8225.

In 2019, Mirsky et al. [57] showed how an attacker can use deep learning to add or remove evidence of medical conditions from a 3D medical scans. They focused on injecting and removing evidence of lung cancer from CT scans (they chose lung cancer because it is common and has the highest mortality rate). In order to modify the file containing the 3D medical scans, the attacker must have access to it. One attack vector they explore is remote infiltration through the Internet. They use Shodan to identify 1,849 medical image servers and 842 medical picture archiving systems that are addressable to anyone who has access to the Internet. They believe the security of health-care systems lags behind modern security standards. They also note that even if there are no exploitable vulnerabilities on these devices, simply knowing they exist presents opportunities for social engineering attacks and physical access attacks.

Censys

In 2015, Adrian et al. [14] used a modified ZMap to identify devices vulnerable to the Logjam flaw in export grade Diffie-Hellman. They found that 8.4% of the Alexa Top 1M domains were potentially vulnerable to the flaw.

In 2015, Springall et al. [69] presented a comprehensive analysis on how FTP is abused. Using Censys, they identified 13.8M IPv4 addresses hosting FTP servers. 1.1M (8%) of these servers permit anonymous (no credentials required) login. They created an FTP enumerator and collected over 600M files and directories — including password databases, private keys, personal photographs and financial information (they infer this from file names and extensions) — from the anonymous FTP servers. Out of the 1.1M anonymous FTP servers, more than 20K allowed an anonymous user to write data. They found several malicious campaigns using the anonymous write permissions to distribute malware and launch DDoS attacks.

In 2016, Li et al. [53] used Censys to identify 46K industrial control systems or ICSs (defined in Section 3.4 on Page 64) that were publicly accessible. They see this as an issue as the ICSs were never designed to be publicly accessible on the Internet. The ICSs lack basic security features such as encryption and authentication and therefore are inherently vulnerable.

In 2017, Krombholz et al. [49] published a paper exploring why it has been so hard to deploy TLS correctly and to study the usability of deploying HTTPS. They used Censys to show that 20,890,000 websites were using Apache.

In 2019, Kotzias et al. [48] used Censys' data to retrieve raw protocol banners from HTTP(S), POP(S), IMAP(S), SMTP, SSH and FTP scans. They parsed the banners for application names and versions then searched the National Vulnerable Database for exploits that potentially affect those applications with that version.

In 2019, Kumar et al. [50] used Censys to identify vulnerable home routers.

In 2020, Leurent et al. [52] report the first practical implementation of the SHA-1 collision attack and they used Censys to show 17% of devices use SHA-1 for signatures and 9% of devices use HMAC-SHA-1.

2.7.5 Analyzing TLS Certificates

Censys

In 2016, Kumar et al. [51] used Censys to analyze how well certificate authorities construct certificates. They conclude that 0.02% of certificates have errors.

Int 2016, Springall et al. [70] used Censys to identify devices using Diffie-Hellman and Elliptic Curve Diffie-Hellman with reused private values, TLS session resumption and TLS session tickets.

In 2016, VanderSloot et al. [71] combined data from Censys and CT logs. They examined all domains in .com, .net and .org. They find that combining Censys with CT logs finds the majority of certificates.

In 2017, Kim et al. [45] used Censys to identify 122 code signing certificates used for TLS by searching for keywords that explicitly indicate code signing usage.

2.7.6 Analyzing Content Filters on the Internet

Shodan

In 2013, Daleket al. [31] presented a methodology for identifying installations of URL filtering products in Internet service providers around the world. They searched Shodan for commonly appearing keywords and headers for URL filtering products: “Blue Coat”, “McAfee SmartFilter”, “Netsweeper” and “Websense”.

Censys

In 2020, Raman et al. [63] downloaded the Censys HTTP and HTTPS measurement data on September 12, 2019. They found content filters in 154 countries probed by Censys. They note that Censys was not designed to measure censorship and Censys does not request content that is commonly censored when it is conducting Internet-wide scanning. The authors are unclear on exactly what

they mean by this, we assume that because Censys indexes very few Shallow Web resources that it may not be detected by some filtering systems.

2.8 Conclusion

In this chapter, first we provided a new definition for the Surface, Shallow and Deep Internet that do not depend on how much of the Internet a conventional search engine can index. We partitioned our definitions into distinct categories with the intent that future protocols (such as HTTPS2 or a new protocol that has not been created at the time of writing this thesis) may be classified based on our definitions. It is our expectation that these definitions allow us to use the terms Surface and Deep Internet without the meaning of these terms constantly changing.

We proposed a new definition of a search engine to include the Surface Internet, Shallow Internet and Deep Internet. Our definitions are based on how some authors [26, 40, 67, 74] define the Surface Web and Deep Web. We chose to not use the word "Web" in our definitions to emphasize that our definitions apply to more protocols than HTTP and HTTPS.

We introduce the term SERICE and clearly distinguish a SERICE from a Web search engine and give two examples Shodan and Censys. We provided examples of how Shodan and Censys are being used by researchers to discover malware, measure the Internet, analyze certificates, analyze content filters. SERICEs primarily target properties while Web search engines primarily target resources. There is a possibility of future SERICEs supporting resource searching, we discuss this further in the future work section of this thesis (Section 4.3 Page 72) by introducing the concept of an *Internet search engine*.

The main takeaway of this chapter is SERICEs are search engines that crawl the Surface Internet and a small amount of the Shallow Internet across many protocols. Web search engines are search engines that crawl the Shallow Internet but only on HTTP and HTTPS. We propose a definition for the concept of a search engine that crawls across the Surface Internet and Shallow Internet for many protocols. We would classify such a search engine as an "Internet search engine".

Chapter 3

Empirical Scanning Analysis of Shodan and Censys

In our empirical scanning analysis we wish to determine how much resources are consumed by Shodan and Censys, how fresh the results obtained from Shodan and Censys are, and how does the operation of Shodan and Censys differ from the perspective of the network edge. To do this we ask and answer the following seven questions:

Q1 How fast do Shodan and Censys update after a Responding Internet-Connected Entity (RICE) changes a service's banner? (Section 3.2.1 on Page 40)

Q2 How much traffic can a RICE expect to receive from Shodan and Censys? (Section 3.2.2 on Page 44)

Q3 Which services do Shodan and Censys most frequently scan? (Section 3.2.3 on Page 48)

Q4 How many unique IP addresses do Shodan and Censys use for scanning? (Section 3.2.4 on Page 52)

Q5 Do Shodan's and Censys' scanners located geographically near to a RICE scan it more frequently than further away scanners? (Section 3.2.5 on Page 54)

Q6 What are the scanning patterns of Shodan and Censys? (Section 3.2.6 on Page 57)

Q7 What level of the Surface and Deep Internet do Shodan and Censys index? (Section 3.2.7 on Page 61)

We provide a methodology for each question and discuss our results.

3.1 Methodology

In order to answer these questions, we designed an experiment where we setup five RICEs (using virtual machines) around the world and analyzed traffic received from Shodan and Censys.

3.1.1 Setting Up Our Virtual Machines

We provisioned five virtual machines (VM) located in San Jose, Tokyo, Montreal, Paris and Sao Paulo. More details of these VMs can be found in Table 3.2. We used Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances to host our VMs. Each EC2 instance is running Amazon Linux 2 Amazon Machine Images (AMI) running Linux kernel 4.14 64bit(x86) operating system. Each instance was launched on February 19th, 2020 with firewall rules set to block all traffic except SSH from our lab computer. Each VM was assigned a static public IP address that remained constant for the duration of our experiment. We then configured a service for HTTP, HTTPS, SSH, SMTP and FTP on the VMs as described in Subsection 3.1.2. As of March 10th, 2020, we allowed traffic from any address to reach the VMs. On March 10th, 2020, each VM was configured with the same firewall rule set which can be found in Table 3.1. Figure 3.1 shows a graphical representation of the locations of our VMs and Censys' scanning location. We point out that all of Censys' scans come from a single location: Ann Arbor, Michigan, USA. Shodan scans from multiple locations, described in Subsection 3.2.5.

Table 3.1: Virtual Machine Firewall Rules

Direction	Transport Layer	Port Range	Source IP Address
Inbound	Any TCP	Any	Any
Outbound	Any	Any	Any

Table 3.2: Location and IP Address of Virtual Machines

City	AWS Region	Private IP Address	Public IP Address
San Jose	US West (N. California)	172.31.6.99	54.219.179.30
Tokyo	Asia Pacific	172.31.39.155	52.197.29.146
Montreal	Canada (Central)	172.31.26.149	35.183.157.209
Paris	Europe	172.31.26.176	15.188.183.150
Sao Paulo	South America	172.31.13.62	18.229.178.20

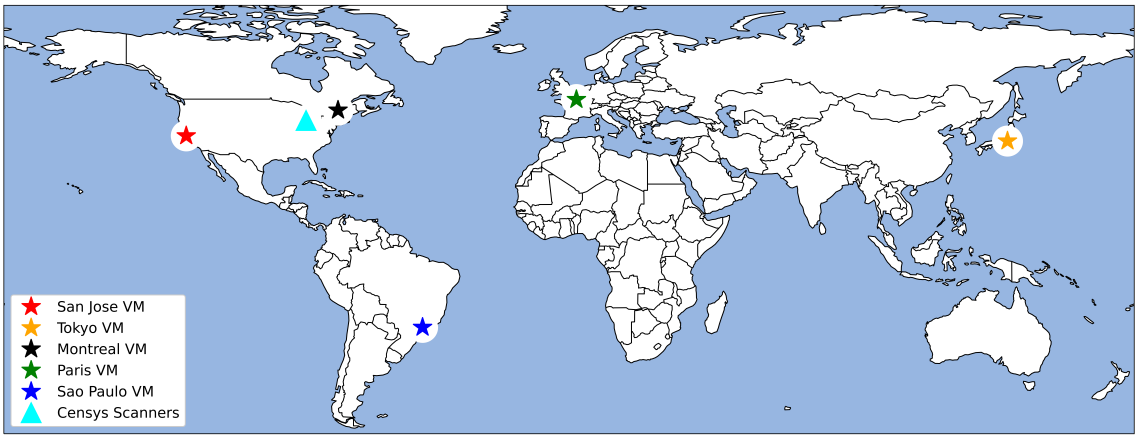


Figure 3.1: World map showing our VMs (stars) and the geographic location that all Censys TCP Sessions came from (triangle).

3.1.2 Configuring Software on Our Virtual Machines

Once each VM was running and the firewalls configured we installed five services to receive port scans on SSH, HTTP, HTTPS, FTP and SMTP. Table 3.3 shows which software we used for each service.

Each VM is running "tcpdump" to collect all network traffic as packet captures (pcaps). We analyze these pcaps to determine when Shodan and Censys have scanned our VMs. The following command was used to collect packets on the VMs.

```
tcpdump -i any -G 86400 -K -n -w %B%d.pcap
```

Table 3.3: Services Running on Our Virtual Machine

Service	Software	Port
FTP	vsftpd	21
SSH	Open SSH	22
SMTP	postfix	25
HTTP	Apache	80
HTTPS	Apache	443

The purpose of these flags is:

- "-i any" - Capture packets from all interfaces.
- "-G 86400" - Number of seconds to wait before starting a new dump file. 86400 seconds is 24 hours.
- "-K" - Don't verify TCP checksums. This will allow us to record packets that have invalid checksums.
- "-n" - Don't convert host addresses or port numbers to names.
- "-w %B%d.pcap" - Write output to pcap file. File name %B%d.pcap uses Linux date(1) formatting to create file names with the current months full name and day such as: "March12.pcap".

We note that during the setup of Tokyo, Montreal, Paris and Sao Paulo, we had a misconfiguration with the SMTP service. Instead of allowing traffic from any IP address, we accidentally configured SMTP to only accept traffic from localhost. This is reflected in our results, and caused many SMTP fields to be left blank as we were only able to collect SMTP data on the San Jose VM. The script we used to configure the VMs can be found in Appendix A.3.

3.1.3 Identifying Foreign IP Addresses

We need to be able to determine which traffic is from Shodan and Censys. We do this by looking at the sender's IP address, and identify if the traffic came from Shodan, Censys or neither.

Identifying Shodan Traffic on November 29th, 2020, we ran a DNS history search for the domain "*.shodan.io" using the DNS history search tool provided by SecurityTrails [7]. 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 addresses 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 the *last seen* date. Appendix A.2 contains a complete list of DNS history of all Shodan subdomains as of November 29th, 2020. For readability, the *first seen* and *last seen* values have been converted from unix timestamps to dates.

Identifying Censys Traffic During our experiment, Censys was scanning the Internet from the "198.108.66.0/23" subnet [2]. This allows us to easily identify Censys traffic, and we assume that all packets whose source IP address matches the subnet "198.108.66.0/23" are from Censys.

3.1.4 Classifying Traffic

We filtered all traffic into three categories: traffic from Shodan, traffic from Censys and other. We used the Python library "pcap-splitter" [6] to parse all of the traffic from Shodan and Censys into TCP sessions, then classified each TCP session as one of the following categories:

SYN: A TCP session containing a single packet with SYN flag set.

SYN_RST: First packet must only have SYN flag set and second packet must have RST flag set.

SYN_RST_EX: First packet must have SYN flag set and second packet must have RST flag set.

SYN_SYN/ACK_ACK: First packet must only have SYN flag set, second packet must only have SYN and ACK flags set, final packet must have RST flag set.

Banner Grab: Any TCP session that reaches the TCP ESTABLISHED state, 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 flag set.

Dangling TCP Session: Any TCP session that contains a packet with an ACK value >1 and contains no packets with the RST or FIN flag set.

We combine these classifications into three sets for the remainder of the experiment: **Closed Port Scan**, **Open Port Scan** and **Banner Grab**.

$$\textbf{Closed Port Scans} = \textbf{SYN\_RST} \cup \textbf{SYN\_RST\_EX}$$
$$\textbf{Open Port Scans} = \textbf{SYN\_SYN/ACK\_ACK} \cup \textbf{Dangling TCP Handshake}$$
$$\textbf{Banner Grabs} = \textbf{Banner Grab} \cup \textbf{Dangling TCP Session}$$

3.2 Results

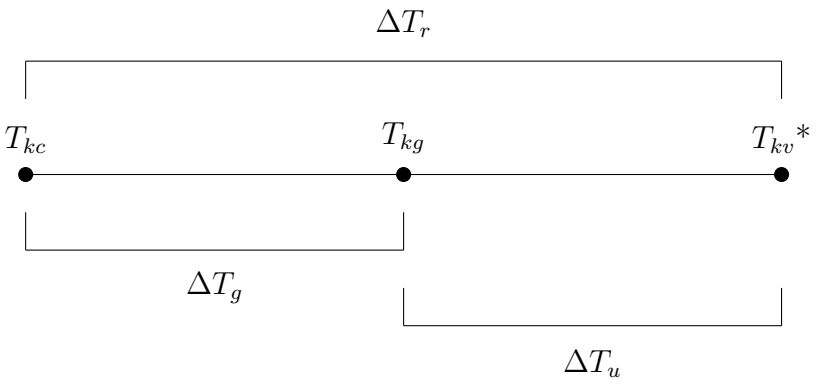
3.2.1 Q1: How Fast do Shodan and Censys Update After a RICE Changes a Service's Banner?

Methodology

Each day, during our experiment (March 10, 2020 to April 26, 2020), we generated a unique 64 character string for each VM. We refer to this as a *key* for the remainder of this thesis. Each day, at approximately 10pm EST, we updated the HTTP service to serve content with a unique key for that day, for that VM. We use the key to calculate three time instances that we call: the *Key Change Instance*, the *Key Grab Instance* and the *Key Visible Instance*. We define these terms below. When querying Shodan and Censys, we searched for the IP addresses of our VMs. It is possible that device-search can make Shodan and Censys pay more attention to our devices. This is out of the scope of this thesis.

Key Change Instance (T_{kc}) Each day a new unique key was generated and written to the *index.html* page on each server. We recorded the unique key and timestamp each time we updated the page. The *Key Change Instance* is the timestamp of when we wrote the new unique key to the *index.html* file.

Figure 3.2: Time Differences Visualisation
* See inline description for notes on resolution accuracy.



$$\Delta T_g = T_{kg} - T_{kc} \tag{3.1}$$
$$\Delta T_u = T_{kv} - T_{kg} \tag{3.2}$$
$$\Delta T_r = T_{kv} - T_{kc} \tag{3.3}$$

Key Grab Instance (T_{kg}) By parsing the pcaps from each server, we are able to find the HTTP GET response packet containing the unique key sent to a SERICE. The *Key Grab Instance* is the timestamp of the packet, containing the unique key, that responds to the banner grab.

Key Visible Instance (T_{kv}) Each day we used the Shodan's and Censys' API to download results for each VM. The *Key Visible Instance* is the timestamp of our request to the SERICE. The API was only queried once every 24 hours. This means the resolution of the "Key Visible Instance" can only serve as an upper bound showing the latest possible time the key could have been made visible.

Time Differences We used the above three time instances to calculate three time durations that we call: *Grab Span* (ΔT_g), *Update Span* (ΔT_u) and *Refresh Span*

(ΔT_r).

Figure 3.2 is a visual representation of the time instances and differences. Since T_{kv} is an upper bound, it will affect ΔT_u and ΔT_r.

We note that these time differences could be applied to Web search engines, where the Change Instance is the time a website is updated, Grab Instance is the time a website is crawled and the Visible Instance is when the SERP reflects the updates.

Results

Figure 3.3 shows a CDF of the amount of time it took Shodan and Censys to banner grab a newly generated key. Figure 3.4 shows a CDF of the amount of time it took Shodan and Censys to make a key visible through their API after banner grabbing that key. Figure 3.5 shows a CDF of the amount of time it took Shodan and Censys to make a key visible through their API after that key was generated.

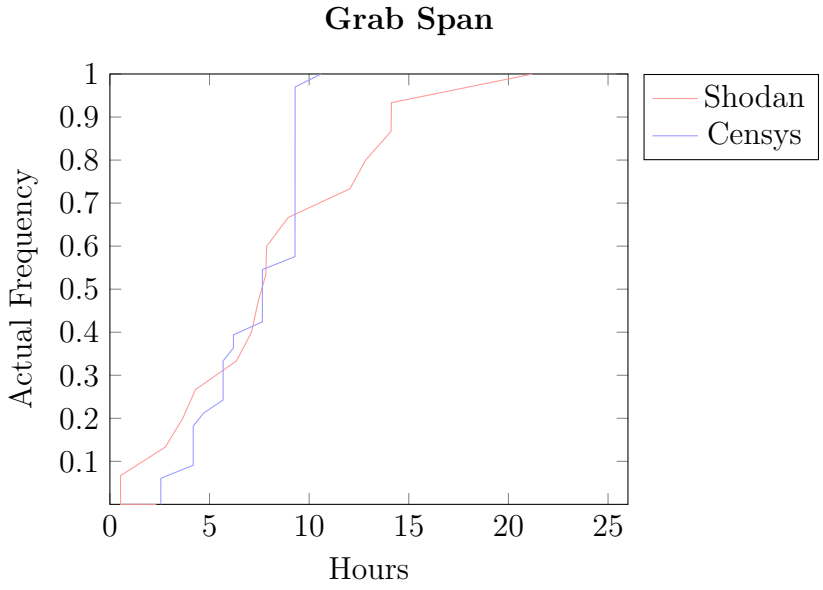


Figure 3.3: A CDF of the number of hours each Shodan and Censys took to banner grab a new key.

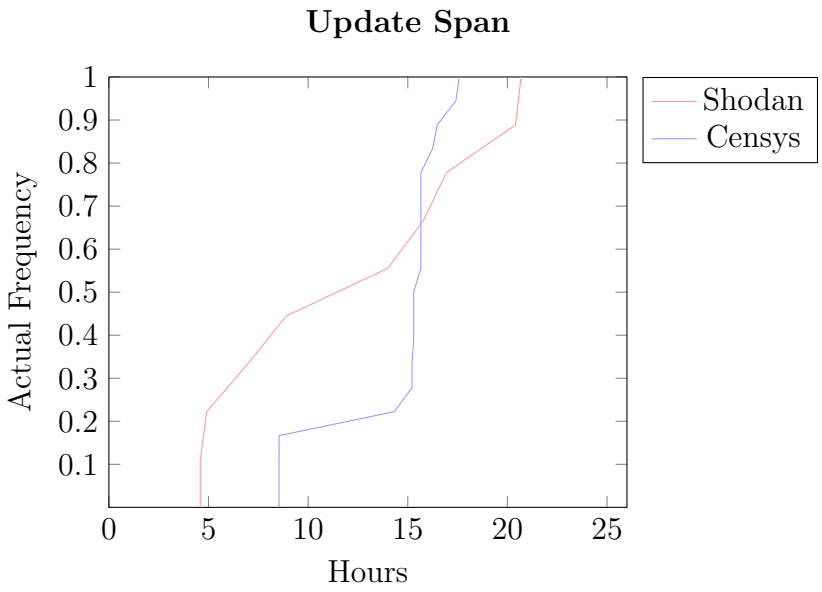


Figure 3.4: A CDF of the amount of time each SERICE took to make a key visible through their API since banner grabbing the key.

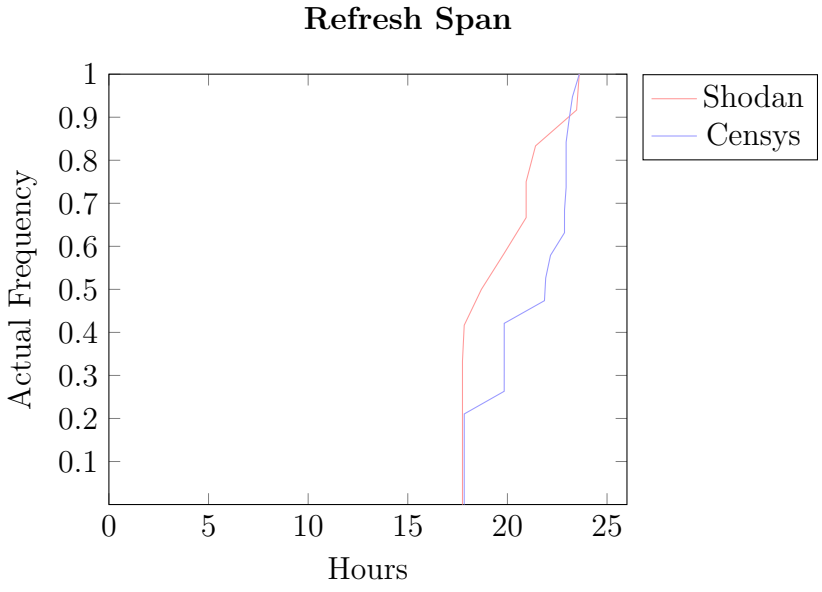


Figure 3.5: A CDF of the amount of time each SERICE took to make a key visible through their API since a new key was generated.

Discussion

Both SERICEs are very fast at refreshing. Shodan and Censys updated every key they found within 24 hours of finding it in a banner grab. If a key was not found

within 21 hours of generating it, then the key was never found by either SERICE. If a key was present in the API but we never saw Shodan or Censys banner grab the key, then this would mean the key must have been banner grabbed from an IP address that we did not identify as Shodan or Censys. All of the keys that were found with both Shodan's and Censys' API had corresponding banner grabs in our pcaps.

3.2.2 Q2: How Much Traffic Can a RICE Expect to Receive from Shodan and Censys?

Methodology

For both SERICEs, we looked at how many: SYN scans (Figure 3.6), banner grabs (Figure 3.7), average SYN scan session duration (Figure 3.8), average banner grab session duration (Figure 3.9), standard deviation of hours between the start of each SYN scan (Figure 3.10), and standard deviation of hours between the start of each banner grab (Figure 3.11).

Results

Calculating the standard deviation of time between SYN scans (Figure 3.10) and banner grabs (Figure 3.11) requires at least two data points. Figures 3.6 and 3.7 represent a single scan or banner grab which has a single timestamp representing the time the scan or banner grab was received. Figures 3.10 and 3.11 show the standard deviation of the time between scans or banner grabs. This means, in order to have a data point for a given VM and protocol, there must be a corresponding value >2 in Figures 3.6 and 3.7. This is why in Figures 3.10 and 3.11 some entries are missing that were present in Figures 3.6 and 3.7.

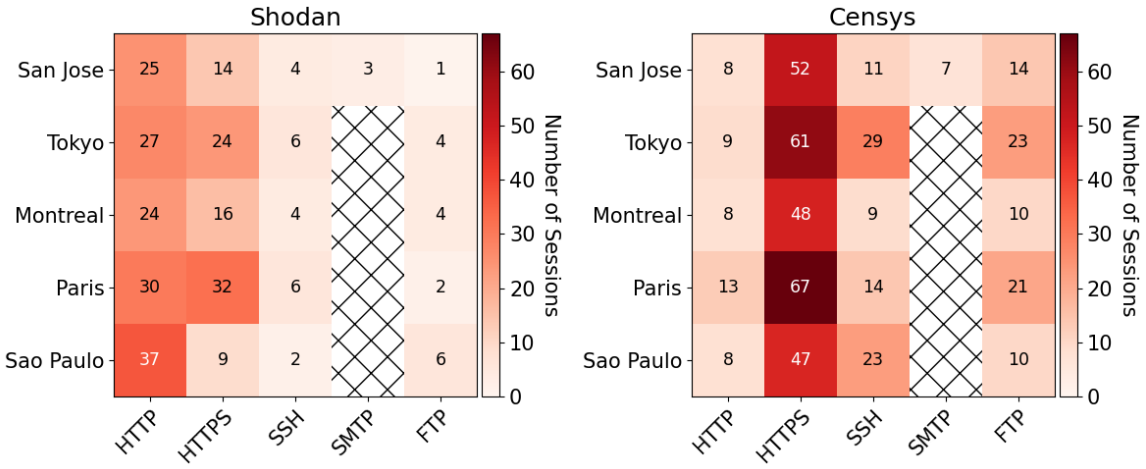


Figure 3.6: The number of SYN scans for each service on each VM.

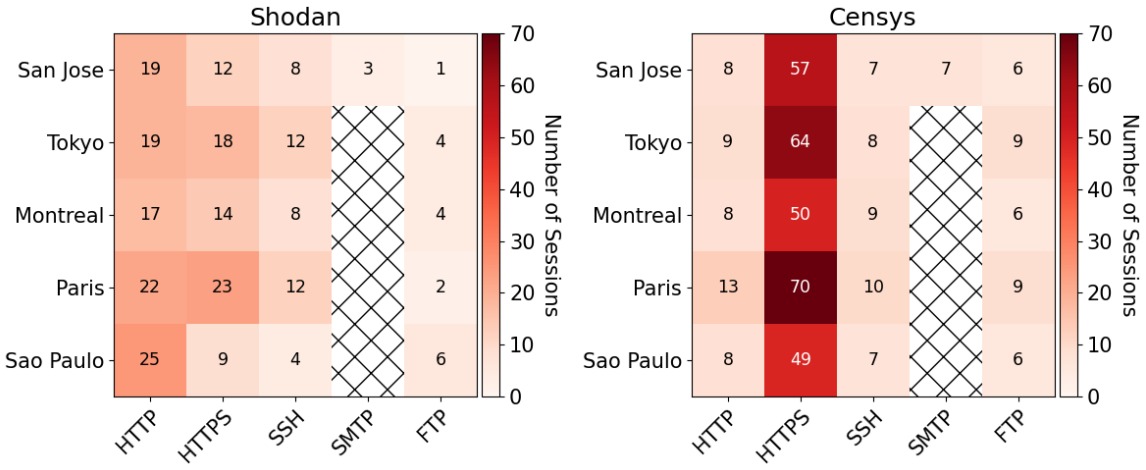


Figure 3.7: The number of banner grabs for each service on each VM.

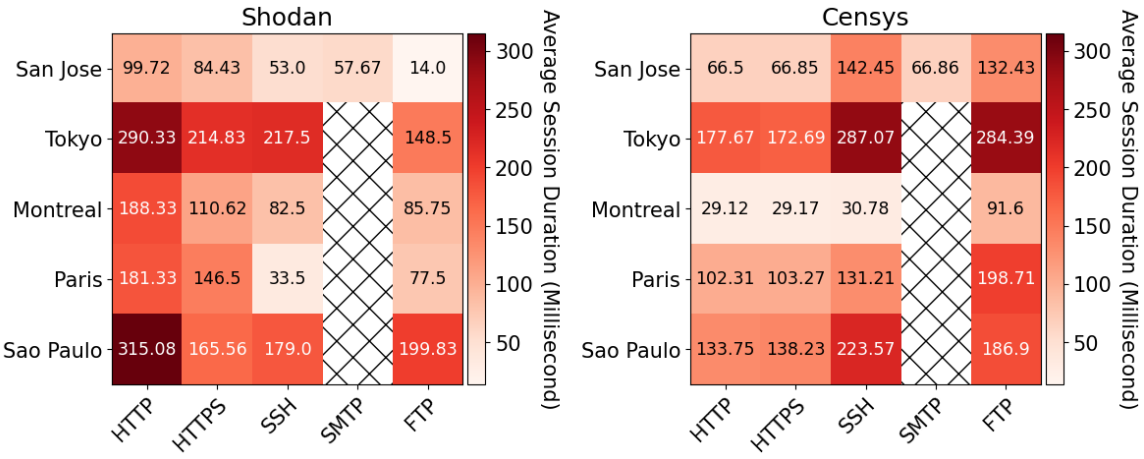


Figure 3.8: Average SYN scan session duration in milliseconds for each service on each VM.

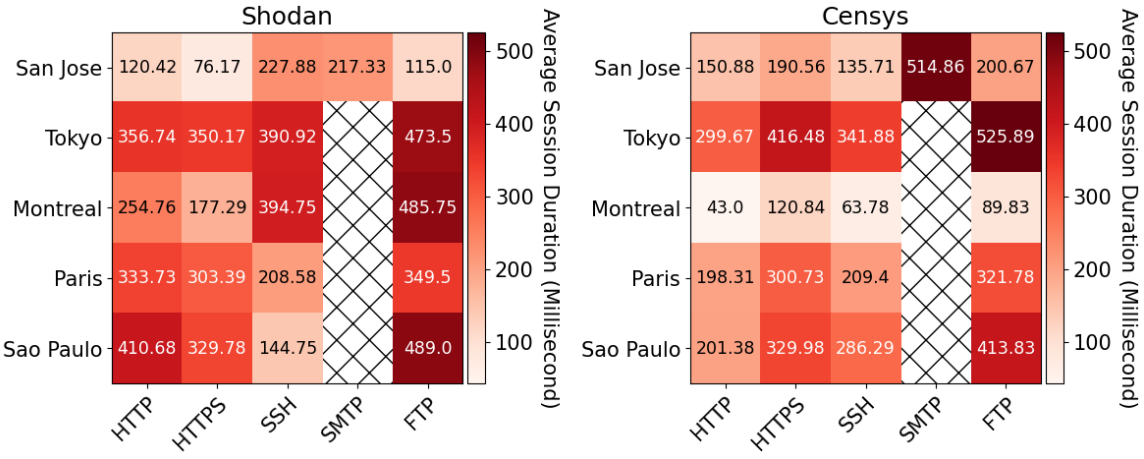


Figure 3.9: Average banner grab session duration in milliseconds for each service on each VM.

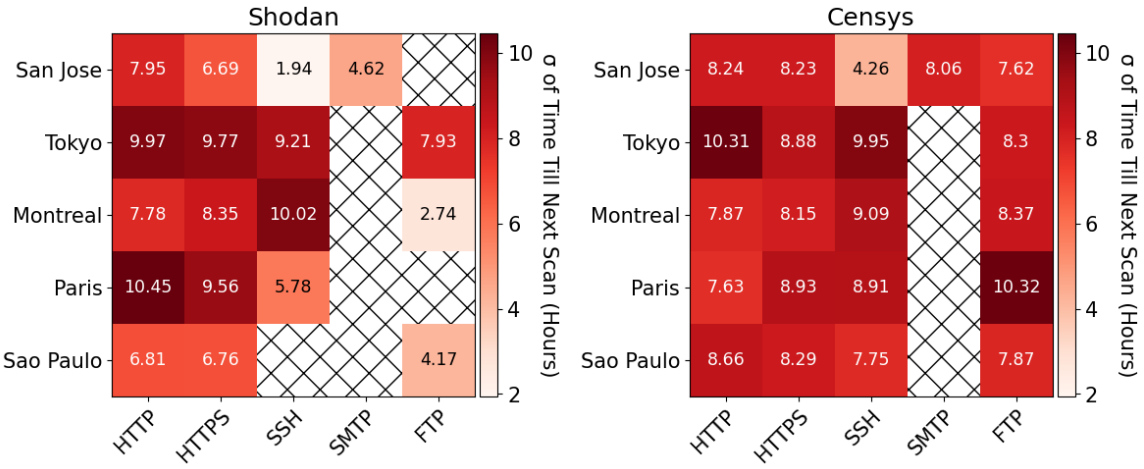


Figure 3.10: The standard deviation (σ) of the number of hours between the start of each SYN scan.

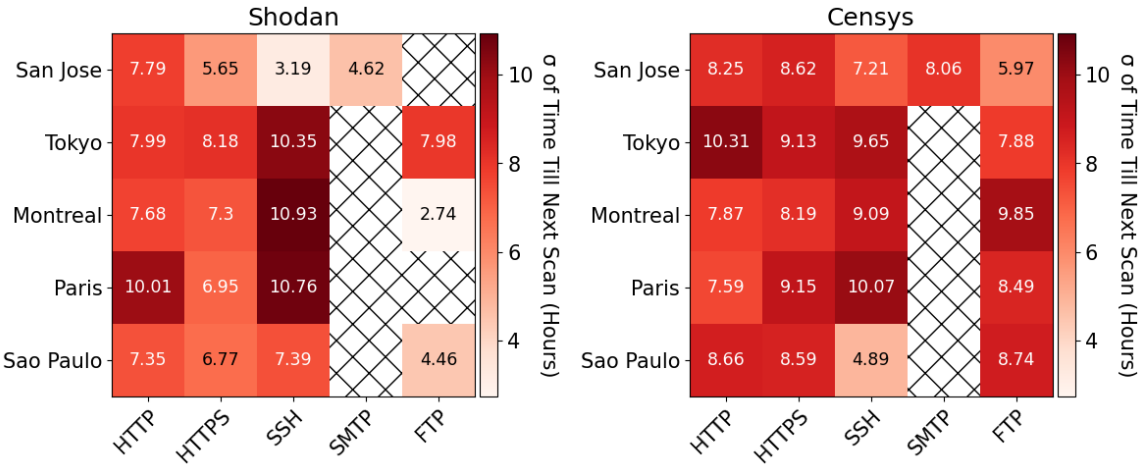


Figure 3.11: The standard deviation (σ) of the number of hours between the start of each banner grab.

Discussion

We see Shodan did SYN scans and banner grabs primarily from HTTP and HTTPs services. We speculate this could be due to us updating the key on the website. It is possible that the SERICEs noticed the frequent changes on HTTP and conducted more banner grabs. Censys has a heavy focus on HTTP SYN scans and banner grabs. We speculate this could be due to Censys' focus on TLS certificates. Censys also performed the same number of HTTP SYN scans as banner grabs. Shodan performed the same number of FTP SYN scans as banner grabs. It is unclear why this is the case. Both SERICEs perform most banner grabs in under three seconds and SYN scans in a fraction of a second. We believe that these session durations and the resources Shodan and Censys consume from a scanned RICE would be commonly acceptable, from the RICE's point of view, in practice, but this ultimately depends on the application and the RICE's available resources.

3.2.3 Q3: Which Services do Shodan and Censys Most Frequently Scan?

Methodology

We examined the five most common ports that received a closed port scan and open port scan. Using the IANA [4] and search results from Shodan and Censys we show possible services Shodan and Censys could be attempting to index with the closed port scans.

Results

Table 3.4 shows the top five open and closed port scans conducted by both SERICEs over the duration of the experiment. We note that many of the closed port scans are not well known ports. In Table 3.5 and Table 3.6 we list the most commonly scanned closed ports next to the IANA registry entry for the port and what Shodan and Censys return, if that port is searched through their Web interface (accessed Nov 30, 2020). Shodan can return multiple results for a single port, in this case we reported the service with the most results.

SERICE	VM	TCP Port(P) Count(C) Most Common → Least Common									
		P	C	P	C	P	C	P	C	P	C
Top Five Closed Port Scans											
Shodan	San Jose	444	12	1177	11	81	10	5555	9	1604	9
	Tokyo	1177	17	7443	14	53	14	81	14	5800	13
	Montreal	1177	13	81	11	4782	11	82	10	1723	10
	Paris	81	19	444	16	2086	15	102	14	2222	14
	Sao Paulo	3460	12	82	11	6666	11	53	10	51235	10
Censys	San Jose	5900	14	9200	13	83	13	2082	12	4567	12
	Tokyo	9200	19	5902	18	8088	18	8080	17	5901	17
	Montreal	9200	13	2082	12	8081	11	8088	11	16993	11
	Paris	9200	19	5902	19	8090	18	83	18	5900	18
	Sao Paulo	9200	13	2083	13	8081	13	88	12	8090	12
Top Five Open Port Scans											
Shodan	San Jose	80	25	443	14	22	4	25	3	111	2
	Tokyo	80	27	443	24	22	6	21	4	111	4
	Montreal	80	24	443	16	111	6	21	4	22	4
	Paris	443	32	80	30	22	6	111	4	21	2
	Sao Paulo	80	37	443	9	21	6	111	4	22	2
Censys	San Jose	443	52	21	14	22	11	80	8	25	7
	Tokyo	443	61	22	29	21	23	80	9		
	Montreal	443	48	21	10	22	9	80	8		
	Paris	443	67	21	21	22	14	80	13		
	Sao Paulo	443	47	22	23	21	10	80	8		

Table 3.4: The top five most common attempted and open port scans received by our VMs from each SERICE.

Port	IANA Registry	Shodan Search Result
53	DNS	-
81	-	HTTP
82	xfer	XtremeRAT
102	iso-tsap	multiple
444	Simple Network Paging Protocol	SonicWALL Firewall HTTP Config
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 Firewall
51235	-	rippled

Table 3.5: List of ports Shodan scanned that were not part of our experiment and which services are associated with each port. IANA and Shodan accessed November 29, 2020.

Port	IANA Registry	Censys Search Result
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 3.6: List of ports Censys scanned that were not part of our experiment and which services are associated with each port. IANA and Censys accessed November 29, 2020.

Discussion

We believe that the number of closed port scans indicated that Shodan and Censys have a high interest in discovering services on those ports. Several of Shodan's closed port scans are displayed on their website as HTTP or SSH services. This suggests Shodan may be sending requests to ports for multiple services, storing any banner returned, and determining what the service running on that port is by processing the banner. Censys, however, has very few of the closed ports returning results through the search interface. This suggests Censys could be looking to support these services in the future, or the Censys infrastructure may be being used for private research. This could include work that will be made public, but is currently in progress. Censys heavily scans port 443 and scans the most popular port on all five VMs almost twice as much as Shodan. We believe this could be due to Censys focus on HTTPS.

3.2.4 Q4: How Many Unique IP Addresses do Shodan and Censys Use for Scanning?

Methodology

We are interested in the number of IP addresses each SERICE uses to conduct scanning. The number of IP addresses used may indicate the number of scanners Shodan and Censys use. These numbers are calculated by taking every TCP session from Shodan and Censys and counting the number of unique IP address-port pairs. The total for all five services is calculated by counting the number of unique IP addresses from each SERICE.

Results

Table 3.7 shows the number of unique IP addresses used by Shodan and Table 3.8 shows the number of unique IP addresses used by Censys. The *All Five Services* column shows the total number of unique IP addresses seen from each SERICE by that VM. It is not a sum of the other columns. Similarly, the *All VMs* row represents the total number of unique IP addresses seen for a given protocol.

VM	HTTP	HTTPS	SSH	SMTP	FTP	All Five Services
San Jose	3	3	4	3	1	10
Tokyo	3	6	4	3	3	14
Montreal	5	3	3	3	4	9
Paris	4	7	4	4	2	12
Sao Paulo	5	1	2	5	5	13
All VMs	11	10	10	11	12	16

Table 3.7: Total number of unique IP addresses, used by Shodan, that sent traffic to our VMs.

VM	HTTP	HTTPS	SSH	SMTP	FTP	All Five Services
San Jose	13	51	14	12	13	73
Tokyo	9	46	17	7	14	71
Montreal	12	51	14	7	11	72
Paris	15	52	15	7	11	75
Sao Paulo	13	49	20	7	12	77
All VMs	35	146	51	38	39	186

Table 3.8: Total number of unique IP addresses, used by Censys, that sent traffic to our VMs.

Discussion

Shodan's API queries return a "crawler" field which contains a 40 character hexadecimal string. We took all IP addresses from Shodan and calculated their SHA1 hash. Each hash appears in the "crawler" field in at least one result in Shodan's API query. This suggests that a single IP address may correlate to a single scanner. In total, our VMs observed 16 unique Shodan IPs. This is consistent with the 16 unique crawler IDs in the API query results.

We observed 11.6 times more IP addresses used by Censys, however, Censys only scans from a single location. We note the very high number of IP addresses dedicated to HTTPS scanning. This reflects Censys' focus on HTTPS indexing.

3.2.5 Q5: Do Shodan's and Censys' Scanners Located Geographically Near to a RICE Scan it More Frequently than Further Away Scanners?

Methodology

In order to determine if a SERICE would use scanners located closer to a RICE we needed to compare two values. The average distance (AD_v) of the RICE from each scanner and the observed distance (OD_v) from each scanner. We define those values as follows:

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.

$$AD_v = \sum_{s \in S} \frac{\text{distance}(s, v)}{|S|} \tag{3.4}$$

$$OD_v = \sum_{k \in K_v} \frac{\text{distance}(F(k), v)}{|K_v|} \tag{3.5}$$

All geographic data was obtained by querying WhoisXMLAPI [11] accessed September 12, 2020. World maps were generated using the Python library Cartopy 0.18.0 [1].

Results

Table 3.9 shows the average distance (AD_{vm}) and the observed distance (OD_{vm}) from all scanners to each VM. Figure 3.12 shows which country the scanner was located in that sent how many closed port scans, open port scans and banner grabs. Figure 3.13 shows the number of TCP sessions used by scanners based on their geographic location.

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 3.9: The average distance from all Shodan scanners to each VM (average distance (AD_{vm})) and the average distance between the endpoints of each TCP Session (observed distance (OD_{vm})).

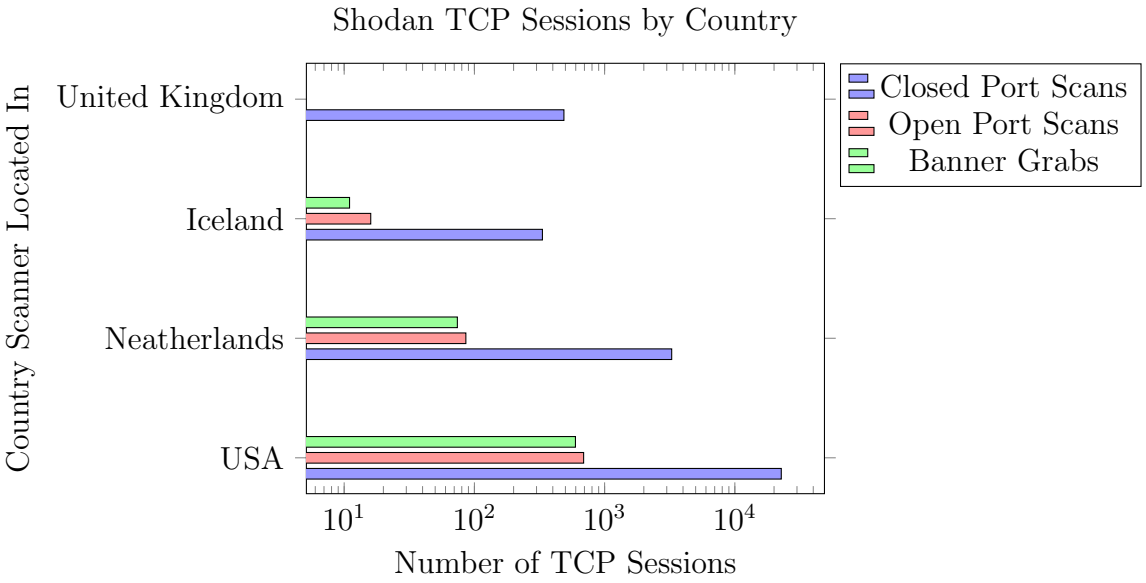


Figure 3.12: Number of TCP Sessions from a Shodan scanner by country the scanner is located in.

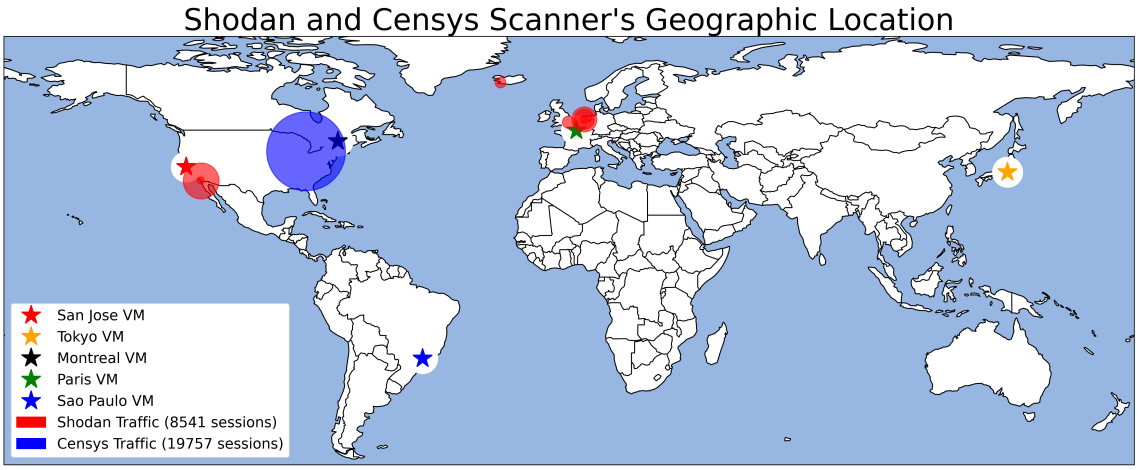


Figure 3.13: World map showing the geographic location of Shodan and Censys scanners. Area of circle is proportional to the number of TCP Sessions that location had with our VMs.

Discussion

We noticed the Montreal VM is getting a disproportional amount of traffic from Shodan scanners located close to the machine. We can see scanners in the US are responsible for the majority of the TCP sessions. The UK is only responsible for closed port scans. This suggests that Shodan is only scanning ports from the UK that were not part of our experiment.

Unlike Censys, Shodan's scanners are found in several countries. Wan et al. [72] 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 traffic missed and three origins further reduces the miss rate to 1%. Since Shodan scans from several origins it follows that Shodan's coverage is better than Censys'.

3.2.6 Q6: What are the Scanning Patterns of Shodan and Censys?

Methodology

We compare the number of TCP Sessions received from Shodan and Censys scanners to the number of distinct ports used by those scanners. We uniquely identified each scanner by its IP Address. We took the set of all TCP Sessions from each scanner and counted the number of unique VMs visited. For example if a scanner sent 1000 HTTP port scans, 1 HTTPS scan, 10 SSH scans, and 1 SSH banner grab the total number of distinct ports is 3.

Results

Figures 3.14 and 3.15 show the number of scanners that connect to how many distinct ports on our VMs. Figures 3.16 and 3.17 show the number of distinct ports used for open port scans and banner grabs. Figure 3.18 shows the percent of scanners that connected to how many of our VMs.

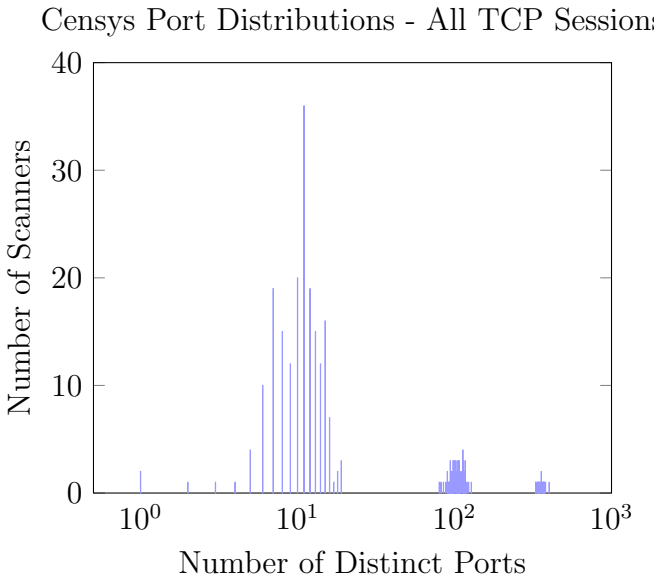


Figure 3.14: Number of scanners that connect to how many distinct ports on our VMs.

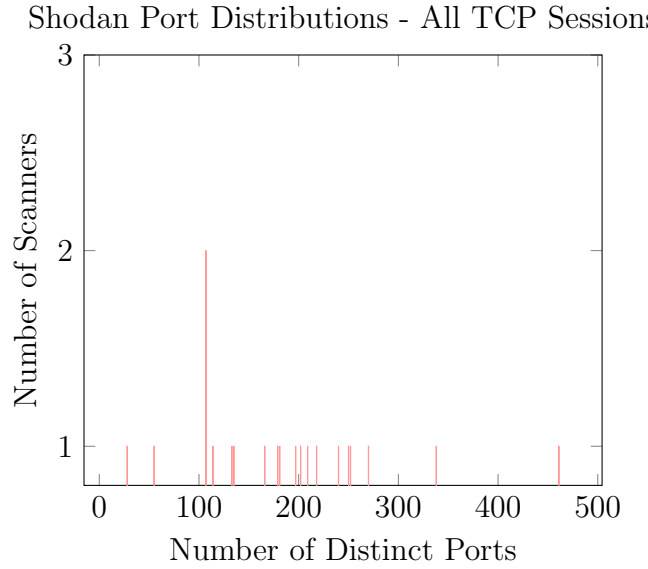


Figure 3.15: Number of scanners that connect to how many distinct ports on our VMs.

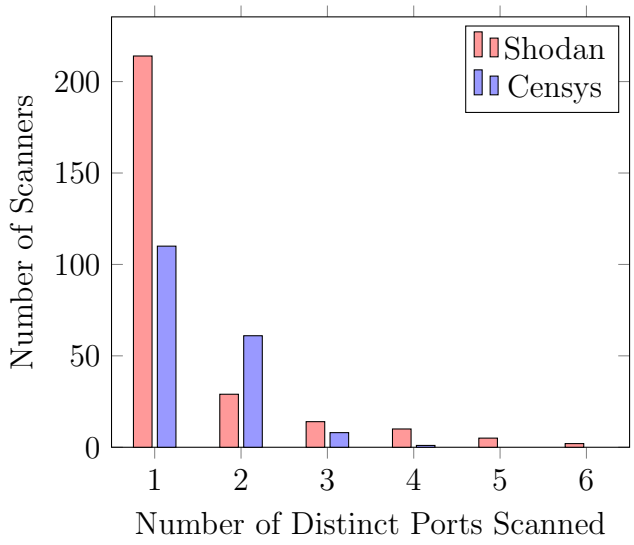


Figure 3.16: Number of distinct ports used for open port scans

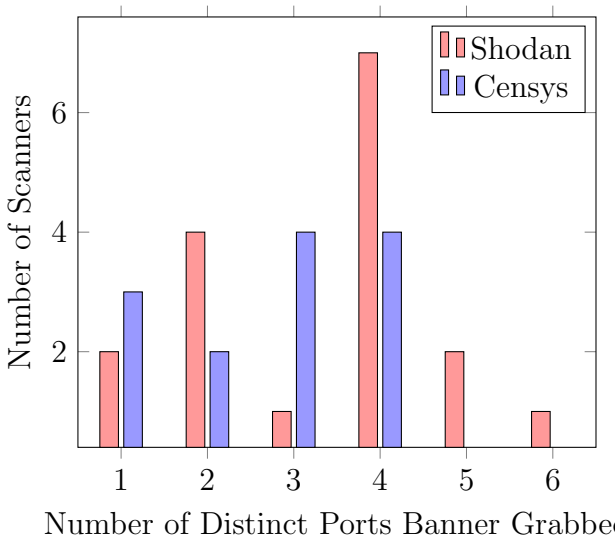


Figure 3.17: Number of distinct ports used for banner grabbing

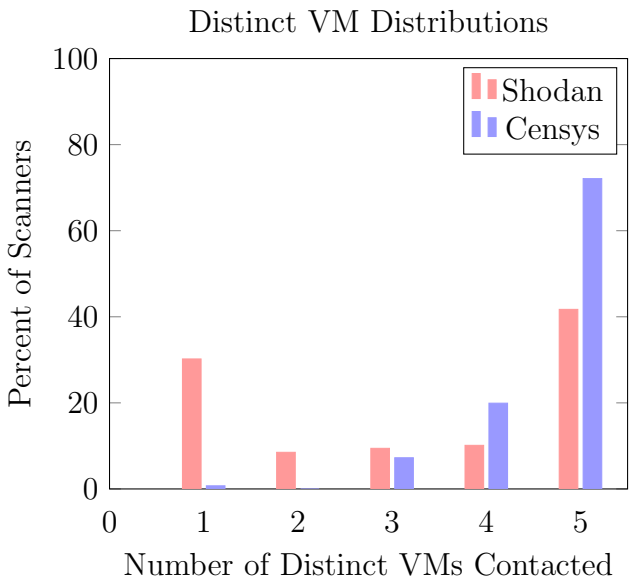


Figure 3.18: Percent of scanners that contacted a distinct number of VMs during our experiment.

Discussion

Figure 3.14 shows that Censys has many scanners that are responsible for scanning the same number of ports. 71% of Censys scanners are responsible for less than 20 ports and only 0.7% scan a single port. Figure 3.15 shows that Shodan has only two scanners that scan the same number of ports. We observed all but two of Shodan's scanners scanning over 100 ports. We observed the majority of Censys' scanners scanning between 8 and 14 ports. This implies that Censys' scanners are very similar to each other while Shodan's scanners vary widely.

Figures 3.16 and 3.17 imply that both Shodan and Censys port scan very few ports across a large number of scanners but once they conduct banner grabs across multiple ports with very few scanners. We speculate this is due to horizontal scans being used during port scans to reduce the risk of the scan being blocked. Once the RICEs have been identified with the port scan a banner grab is issued only to IP addresses found durring the horizontal scan.

Figure 3.18 shows approximately 75% of Censys scanners contacted all five of our VMs. Shodan's only had 40% contact all of our VMs, 30% of Shodan's scanners only contacted a single VM.

Further analysis of Shodan and Censys scanning patterns may lead to identifying unique patterns that would allow us to distinguish Shodan and Censys traffic that originated from an IP address that was not found by our methods. Some variables that we believe may be useful to use for distinguishing traffic are: time of day, ports scanned, number of ports scanned by a unique IP address, and if multiple RICEs are analyzed we could examine the number of RICEs that were scanned near the same time (Internet-wide IPv4 port scanning can be accomplished in under 45 minutes).

3.2.7 Q7: What Level of the Surface and Deep Internet do Shodan and Censys Index?

Methodology

We examine the HTTP GET requests from Shodan and Censys to see what resources are being requested from our HTTP service. We combined HTTP GET requests from

all VMs, as we are only interested in which level of the Surface and Deep Internet Shodan and Censys are scanning.

Results

Censys only used one user agent, "Mozilla/5.0 zgrab/0.x", for 46 requests and always requests the path "/". Shodan used several user agents (Table 3.10) and requested multiple paths (Table 3.11).

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 3.10: 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 3.11: Shodan HTTP resources requested.

Discussion

Shodan uses five versions of pythons requests library to conduct HTTP banner grabs. The longest user agent claims to be a Windows machine running Google chrome. Perhaps they are simulating a browser environment to attempt to bypass security features that require a browser to have javascript enabled. The user agent field is trivial to spoof, we can only conclude that Shodan's user agents may be inconsistent.

Both Shodan and Censys request the resource located at path "/". This a Default Resource of HTTP and means both index the Default Internet. Shodan also attempts to find resources at four other paths, all are well-known paths that are common across many websites. By our definition this means Shodan indexes the Discoverable Internet.

3.3 Limitations

Misconfiguration of Virtual Machines

The Tokyo, Montreal, Paris and Sao Paulo VMs had misconfigured SMTP services that were configured to listen to connections from IP address "127.0.0.1" instead of the intended IP address "0.0.0.0". This prevented the SMTP service from responding to external requests. This misconfiguration is reflected in our results as we are unable to collect SMTP traffic on 4 of our VMs.

All Virtual Machines Hosted by AWS

It is possible that the known AWS subnet of IP addresses may cause our VMs to receive more port scans than they would if assigned random IP addresses. This is due to Amazon providing a full list of IP ranges that it uses for hosting. This would allow scanners who are specifically targeting Amazon instances to quickly enumerate all of the IP addresses. As of Nov 28th, 2020, a full list of the ranges can be found at `https://ip-ranges.amazonaws.com/ip-ranges.json`.

Diurnal Effects

A diurnal effect, in the context of this thesis, means any change in a result based on time of day. For example, would an Internet-wide scan at 10am return the same results as an Internet-wide scan at 10pm? Any deviation would be considered a diurnal effect. As shown by Durumeric, diurnal effects can affect results of Internet scans [37]. In this thesis we did not consider or correct for diurnal effects.

HTTP is the Only Service with a Key

The scope of this thesis is to collect preliminary results. Censys limits API to 250 queries per month. Each service requires 1 scan per day per VM for a total of 5 scans per day. This means we would need 525 scans to use a unique key to identify when each service was scanned to remain within the free tier of usage. For this reason, we only use a unique key on HTTP.

Blocked Incoming Non-TCP Traffic

We blocked all inbound non-TCP traffic, as it is out of the scope of this thesis. It may be interesting in future work to examine what ICMP traffic or UDP port scans SERICEs are using.

Shodan and Censys Limitations

We acknowledge that it is possible for Shodan to use port scans from IP addresses other than listed in our SecurityTrials [7] query. Similarly, Censys sometimes will make follow-up connections from other machines at dynamic IP addresses [2]. We did not account for these scenarios which means it is possible for some results to be missing.

3.4 Related Work

First we cover two terms used in the related work: *industrial control system* and *programmable logic controller*.

Industrial Control System

An *industrial control system (ICS)* is a general term used for coordinated collections of devices that are used to operate industrial processes. *ICSs* compose the infrastructure required to control motors, pumps, relays, cooling systems, furnaces and many other systems. It is imperative that these devices do not fail as they are responsible for monitoring and controlling critical systems in an industrial environment.

Programmable Logic Controller

Programmable logic controllers (*PLC*) are a type of ICS, designed to continuously monitor the state of of an industrial system. They can be configured to control systems and to raise alarms during system failure.

3.4.1 Identifying SERICEs on the Internet

Chen at al. [25] deployed six honeypots and collected three months of network traffic. They identified Shodan traffic by applying machine learning techniques to the scanning patterns and found 16 scanners similar to Shodan that are not Shodan. We searched for subdomains that matched *.shodan.io to identify Shodan scanners.

Durumeric et al. [56] identify patterns in scanning behaviour and uncover large horizontal scan operations on the Internet. They focus on all scanners that are scanning the Internet, which services are most commonly targeted and discuss the impact of new scanners. It is important to note that some of the scanners they observed will likely be SERICEs. We focus on how two specific SERICEs scan the Internet: Shodan and Censys. They found that a large portion of scanning is targeting services commonly associated with vulnerabilities.

Richter et al. [66] tracked scanning activity of traffic captured at the firewalls of 89k hosts of a major Content Distribution Network.

Heo et al. [43] study network scanning trends from a 31-day-long connection logs 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. A major difference between their study and our study is we only target scans from Shodan and Censys while they are targeting all scanners. They created a methodology for determining if a remote host is a scanner. Once a host has been classified as a scanner they further classify the type of scanning as horizontal, vertical, combined or unclassified. Of the 21 billion connections they identified 2.65 billion scan probes from a total of 3.78 million scanners. Like us, they uniquely identified scanners by their IP addresses. They note that this means they cannot account for IP address spoofing. Heo et al. define a “Responsible Scanner” to be a scanner with an IP address that satisfies at least one of the following conditions:

- The scanner responds to a HTTP GET request containing information that explicitly informs how to be excluded from scanning and states the intention of scanning.

- The subdomain contains the word "scan".

- The second-level label of the domain indicates scanning related or search engine.

- The associated autonomous system indicates scanning intention.

Both Shodan and Censys meet their definition of a responsible scanner. Of the 3.78 million scanners identified only 20 scanners, using a total of 690 unique IP addresses, can be classified as responsible scanners.

3.4.2 Studies that Evaluate or Measure the Functionality of SERICEs

In 2014, Bodenheim et al. [20] investigated the functionality of Shodan. They deployed four Internet connected Allen-Bradley ControlLogix PLCs with static IP address directly accessible via the Internet. All four were deployed in the same subnet. Each PLC exposed two services: HTTP (port 80) and common industrial protocol (port 44818). Shodan does not index port 44818, only port 80. Two PLCs were left with the default configuration. One PLC had its banner obfuscated to the string "KCC02013_$h09mo]" and one PLC had its banner changed to explicitly say it was an Allen-Bradly ControlLogix PLC. All four PLCs received at least one port scan from Shodan in less than 4 days. The HTTP banners were grabbed from all four PLCs within 14 days. Using keywords only, they searched Shodan for the PLCs. Each PLC was found, through Shodan's API, within 19 days of the initial deployment.

One of the major differences between Doenheim et al. and our work is they did not search for their PLCs by IP address. One of the queries they used ("port:80") required them to download and parse 170,467,439 results. Our method only required us to download a single result per VM. However, it is possible that our VMs were indexed much faster because Shodan may give the IP address we provided a higher priority for scanning.

3.4.3 Tools for Processing the Output of SERICEs

Genge et al. [41] developed ShoVAT, a tool which processes the output of Shodan queries and compares it with the National Vulnerability Database [5] to detect vulnerable devices on the Internet. They managed to find 3922 known vulnerabilities across 1501 services [41].

Ercolani et al. [38] used the data Shodan collects about RICEs to create visual models used to attempt to identify which device(s) the RICE is comprised of. Both of these works are similar to ours because they examined how a SERICE represents a RICE. However, our work is only focused on how the data for the RICE was collected and when it was made available to users.

3.5 Conclusion

In this chapter, we answered seven questions. In doing so, we found many similarities and differences between how Shodan's and Censys' scanners operate.

Similarities

We observed both Shodan and Censys detecting changes made to our HTTP services in under 25 hours. This is a very short time to detect a change, if the refresh span of all services is this short, then we would conclude that both Shodan and Censys provide an up-to-date snapshot of the Internet. Since we only tested HTTP, for a short duration of 47 days, we cannot state that this conclusion applies to other services. We observed very short session durations for both Shodan and Censys when conducting port scans and banner grabs. Both Shodan and Censys appear to use horizontal scanning when conducting port scans and both use a small number of scanners to follow up with banner grabs.

Differences

Censys has a large amount of its scans and banner grabs focusing on HTTPS. Shodan attempts to find HTTP(S) and SSH services running on non-standard ports. We observed 16 scanners from Shodan and 186 scanners from Censys. Censys uses many

more IP addresses than Shodan however Shodan uses more geographic locations to conduct scanning from. It may be an interesting question as to which of these methods provides a better snapshot of the Internet.

Discussion

We provided seven methodologies for analyzing a SERICE's scanning patterns and applied them to Shodan and Censys. The scope of our study was limited by the 47 day experiment duration. We used Amazon's predictable IP ranges, did not account for diurnal effects, only analyzed updates on a single service, only analyzed TCP traffic, did not account for scanners that used IP addresses that would bypass our method for identifying Shodan and Censys IP addresses, and our queries to Shodan and Censys have a resolution of one day. We believe our methodologies will provide a more accurate method for analyzing Shodan and Censys if the above limitations are addressed. Over the duration of our experiment, Shodan initiated 8541 TCP sessions (7979 closed port scans, 300 SYN scans and 262 banner grabs) and Censys initiated 19755 TCP sessions (18843 closed port scans, 492 SYN scans and 420 banner grabs).

Chapter 4

Conclusion

In this chapter we return to our claimed contributions, to confirm that they have been met then, we discuss future work, including open problems with Internet-wide scanning (IWS).

4.1 Contributions

In this thesis, we revisit what in our view, are outdated definitions, of the Surface Web and Deep Web and provide our new definitions of the Surface Internet, Shallow Internet and Deep Internet. We believe these definitions are a useful contribution because they do not depend on what a search engine is able to index. The meaning of our terms do not change over time.

We argue that Web search engines primarily target the Shallow Internet, and index resources from HTTP and HTTPS services. We introduce the terms Responding Internet-Connected Entity (RICE) and Search Engine for Responding Internet-Connected Entities (SERICE) and argue that SERICEs primarily target the Surface Internet, indexing RICEs that serve many protocols. We argue that, by our definitions, Shodan and Censys are SERICEs and we summarised how Shodan and Censys are being used in research. We believe our definitions are useful contributions because they allow us to succinctly describe what search engines like Shodan and Censys (by our definition SERICEs) return on their SERPs when a user enters a query.

In our empirical work, we provide seven methodologies for analyzing SERICEs and applied our methodologies to Shodan and Censys. Our methodologies describe how to: identify scans from Shodan and Censys, determine how quickly a change in a RICE is detected by a SERICE, measure how much traffic a RICE receives from a SERICE, determine which services a SERICE scans most frequently, determine how many IP addresses a SERICE conducts IWS from, calculate if a RICE is being

disproportionally scanned by a SERICE's scanners located physically close to them, analyze scanning patterns of a SERICE, and determining which level of the Surface and Deep Internet a SERICE indexes. We believe our methodologies for analyzing SERICEs are useful because Shodan and Censys are commonly used by researchers to: discover RICEs infected with known malware, identify vulnerable RICEs, measure TLS certificates, and analyze content filtering on the Internet. We found that Shodan and Censys conduct most banner grabs in under three seconds and SYN scans in a fraction of a second suggesting that Shodan and Censys provide an up-to-date view of the Internet. Our empirical analysis provides evidence that Shodan and Censys update their search interfaces within 24 hours of conducting a banner grab. Shodan's scanners appear to be scattered geographically around the world while Censys' scanners are all in Michigan, USA. Censys focuses on HTTP/HTTPS more than Shodan and dedicates many scanning resources to these services. The time of day Shodan scans is slightly more consistent than Censys. Our methodologies can be applied by researchers who wish to have confidence that the results of a query to a SERICE are accurate.

4.2 Open Problems With Internet-wide Scanning

4.2.1 Server Name Indication Problem

Transport Layer Security (*TLS*) version 1.2 is defined by RFC5246 [33]. *TLS* is used to provide privacy and data integrity between two communicating hosts. HTTPS traffic is encrypted using *TLS*. We are interested in the TLS extension *Sever Name Indication* (*SNI*), which is defined in RFC3546 [19]. During the handshake of *TLS*, prior to a secure connection being established, an HTTPS client (e.g. Web browser) uses *SNI* to provide the hostname they wish to connect to [19]. This allows for multiple websites to be hosted by a server using one IP address and one port.

IPACs (Section 2.3.2 on Page 22) have no way of knowing the hostname for an IP address they generated. This means an HTTPS server, that requires a user to know the hostname, is inaccessible to an IPAC and therefore cannot be indexed by a SERICE [71]. HTTP services do not have this problem as HTTP does not use TLS.

The SNI problem does not affect Web crawlers. Web crawlers discover new resources by following URLs found on previously indexed resources. URLs may contain the hostname, when this is the case, Web crawlers are able to provide the hostname during the TLS handshake.

We believe the SNI problem could be resolved, at least in part, if IPAC-based scanners (such as SERICEs) conducted Web crawling (in addition to IP address crawling) to discover resources. When the Web crawler finds a URL, it can retrieve the IP address associated with the domain of the URL via DNS. This allows the SERICE to search its database for the newly found IP address, if the IP address matches a previously discovered RICE, the SERICE can add any new information gathered to the index for that RICE.

4.2.2 IPv6 Internet-wide Scanning

It is infeasible to conduct an IWS (Section 1.5 on Page 7) of all IPv6 addresses due to the 128-bit address space of IPv6 [59], because of this, we need a way to generate IPv6 addresses that are in use. However, it is still possible to discover RICEs with the appropriate heuristics. RFC7707 [42] explores several techniques that can be used for generating IPv6 addresses that may identifying a live host. We believe mass adoption of IPv6 could be an issue for IWS as it will be much more difficult to find addresses that identify live hosts.

4.2.3 Network Address Translation with IPv6

Network Address Translation (NAT) is used to reduce the number of public IP addresses a private network uses. Devices behind a NAT are not directly addressable from the public Internet. NAT and IPv6 are designed as solutions for IPv4 address exhaustion (Section 1.5 on Page 3). As more private networks migrate to IPv6, they may assign each device on their network a public IPv6 address instead of using NAT. If that device is vulnerable, then cyber criminals who are conducting IWS will be able to identify the device and potentially compromise it. We believe this could potentially result in the creation of a botnet similar to the Mirai botnet [15].

4.2.4 Dark Web Scanning

This thesis focuses on the Surface and Deep Internet, and we briefly touch on Darknets (Section 2.1.1 Page 9). We believe it is useful to separate the word "Web" from our terms and believe this could also be applied to the Dark Web to create a definition for a *Dark Internet*. We leave the definition of the *Dark Internet* for future work.

The Tor project [9] is a free and open-source browser that allows a user to browse the Tor Darknet. The Tor Darknet is comprised of Tor hidden services [8] which are only reachable through the Tor network using an *onion address*. An onion address is similar to a URL, except it is used to locate hidden services on the Tor Darknet. The goal of hidden services is to make both the user and the server anonymous.

Several Dark Web search engines exist as Tor hidden services. We list some, as well as their onion address, available on the Tor network as of Nov 30, 2020:

1) Kilos available at: dnmugu4755642434.onion

2) Ahmia available at: msydqstlz2kzerdg.onion

3) HayStak available at: haystakvxad7wbk5.onion

4) Torch available at: xmh57jrknzkhv6y3ls3ubitzfqnkrwxhopf5aygthi7d6rplyvk3noyd.onion

5) Onion Land available at: 3bbaaaccczcbdddz.onion

4.3 Future Work

We propose a definition for the concept of a search engine that we call the *Internet Search Engine* (*ISE*). The *ISE* would be capable of indexing resources and RICEs, on the Surface Internet and the Shallow Internet, for many protocols. Similar to how Google's PageRank [23] ranks a webpage based on the number of URLs referencing that webpage, an *ISE* could determine the rank of a resource or RICE based on the URLs discovered from many protocols. For example an SSH banner could have a URL to an HTTPS website, or an FTP server may have a README file with a URL to a website. An *ISE* would be able to answer queries such as: a query for a "3D model of a car" and the *ISE* could respond with a SERP (Section 2.2 on Page 16)

listing URLs of FTP servers that contain files of 3D models of cars (the SERP could have previews of the models next to the URLs), or a query for the title of a videogame where the *ISE* could respond with a list of IP addresses hosting public servers running that game. Crawling URLs from all services will find more URLs than just crawling HTTP and HTTPS. Since there are more URLs from more sources, we believe the ranking algorithm would be able to rank resources better, which may mean the ISE would be able to return higher quality and more relevant resources than a standard Web search engine.

We were unable to find a search engine (that meets our definition of a SERICE) for the Dark Web. However, an online article [28] claims a service named "Ichidan" was a "Shodan for the Dark Web", but is no longer online. We believe it would be possible to create a SERICE that indexes the Dark Web. This is useful because it would allow researchers to quickly answer security-related questions about the Dark Web, without having to spend time and resources designing scanning software for the Dark Web.

Appendix A

A.1 Transmission Control Protocol

Here we review a few well known data networking terms and concepts, as a convenience to readers unfamiliar with networking. These are in addition to the terms at the end of Chapter 1.

Transmission control protocol (TCP) is a transport layer protocol that is used for reliable transmission of packets [61]. *TCP* handles many issues in packet-based messaging including corrupted packets, duplicated packets, packet loss and packets received out of order.

TCP Header

Figure A.1 shows the *TCP header* [61].

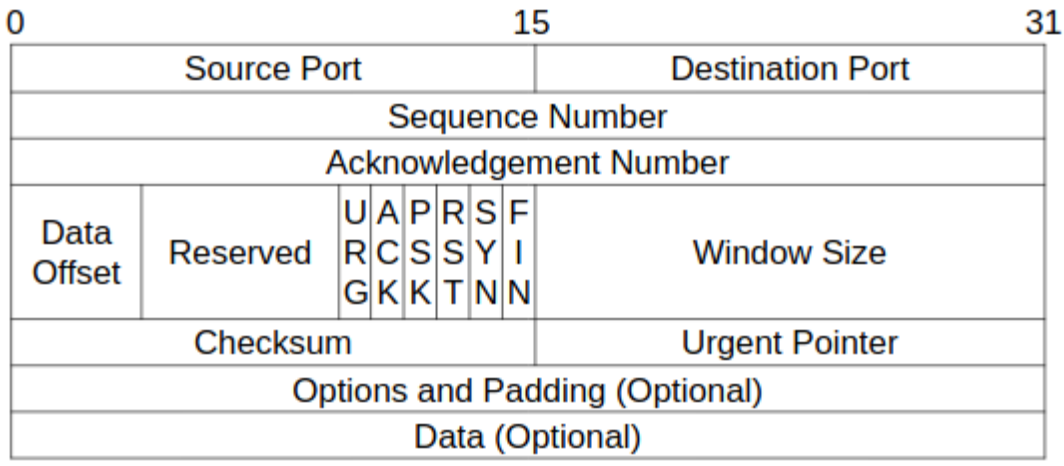


Figure A.1: The TCP header [61]

Source Port is the port number from which the packet originated. (The corresponding source IP address is given in the IP header.)

Destination Port is the port number where the packet is heading. (The corresponding destination IP address is given in the IP header.)

Sequence Number is the initial sequence number for a conversation, if the SYN control bit is set. Otherwise, it is the offset to the first octet of the data segment plus the previous *sequence number*.

Acknowledgement Number represents the next Sequence Number the sender is expecting to receive, if the ACK control bit is set. If the ACK control bit is not set, the acknowledgement number is ignored. Once a TCP connection is in the ESTABLISHED state, the ACK control bit must always be set.

Data Offset the offset to the Data Section of the TCP header.

Reserved for future use. Should always be set to 0.

Control Bits

URG: Urgent Pointer - This is beyond our scope and will not be explained.

ACK: Acknowledgement - Used to communicate that data was received.

PSH: Push Function - Indicates that the receiving host should immediately send this packet to the application without buffering.

RST: Reset Connection - Aborts the connection.

SYN: Synchronize - Sets the sequence number for the connection.

FIN: Finish - Indicates the final packet from sender.

Checksum is used for error-checking of the TCP header.

Options specify several settings. Some options include: window scale, maximum segment size and timestamps. The IANA has defined 254 options [22].

Data is the application layer data.

TCP Port

A *TCP port* is a logical representation of a communication endpoint [64]. Ports are identified by a 16-bit integer which can represent values between 0 and 65535. In the TCP, port 0 cannot be used as it is reserved.

TCP Socket

A *TCP socket* is a logical communication endpoint defined by an IP address and a TCP port number [46]. Each socket can be used for both input and output and an application associates the input and output of network traffic with a socket. This association is known as "binding" [46].

UDP Ports and UDP Socket

UDP ports and UDP sockets exist but are outside the scope of this thesis.

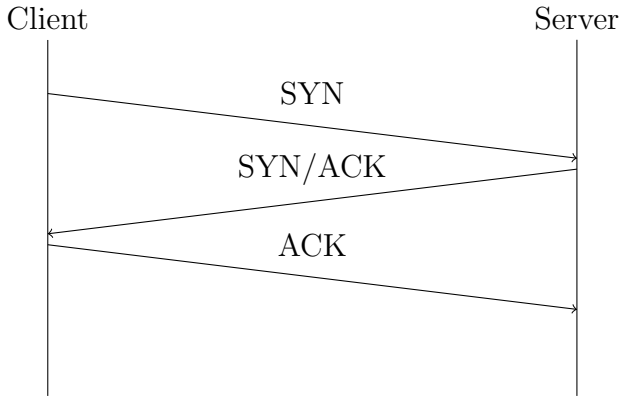
TCP Connection

A *TCP connection* is uniquely identified by a pair of sockets, one for each endpoint [61]. A *TCP connection* may have many states (defined in [61]). In this thesis, we are only interested in ESTABLISHED state. A *TCP connection* is in the ESTABLISHED state when a TCP handshake (see below) has been completed. A *TCP connection* remains in the ESTABLISHED state until either a FIN or a RST is sent or received.

TCP Handshake

A *TCP handshake* consists of a client sending a SYN, which is responded to by the server with a SYN/ACK, causing the server's TCP connection state to become ESTABLISHED. The client then responds to the server with an ACK and the client's TCP connection state is now ESTABLISHED. Figure A.2 is a timing diagram showing the *TCP handshake* between a Client and Server.

Figure A.2: TCP Handshake Timing Diagram



TCP Stream

A *TCP stream* is a series of packets observed on a single node in a network where each packet shares the same TCP connection. A *TCP stream* does not necessarily contain all packets sent from each endpoint. Some packets may follow routes excluding the observed node, packets may have been dropped or packets may have been modified on route.

A.2 Shodan Domains

For the work in Sections 3.1.3 (Page 38) we used SecurityTrails [7] "subdomain" tool (through their API) using the query "shodan.io" to generate a list of Shodan subdomains. For each subdomain, we used SecurityTrails "history" tool (through their API) to query for the historical data associated with that subdomain. All data was accessed Nov 29th, 2020. Table A.1 contains a complete list of all Shodan subdomains returned by SecurityTrails. The *First Seen* and *Last Seen* columns have been converted from a Unix timestamp to a date. The *First Seen* column is the timestamp of when SecurityTrails became aware of the domain name being associated with the IP address, and the *Last Seen* column is when SecurityTrails became aware the domain was no longer registered to that IP address. Table A.1 spans multiple pages from Page 78 to Page 84. This data is used during our empirical study to identify IP addresses used by Shodan scanners (Section 3.1.2 on Page 37).

Subdomain	IP	First Seen	Last Seen
beta.shodan.io	172.67.75.154	2020-06-03	2020-11-27
beta.shodan.io	104.26.9.142	2020-06-03	2020-11-27
beta.shodan.io	104.26.8.142	2020-06-03	2020-11-27
beta.shodan.io	104.26.9.142	2020-01-24	2020-06-03
beta.shodan.io	104.26.8.142	2020-01-24	2020-06-03
beta.shodan.io	104.27.75.9	2020-01-20	2020-01-24
beta.shodan.io	104.27.74.9	2020-01-20	2020-01-24
account.shodan.io	172.67.75.154	2020-06-02	2020-11-27
account.shodan.io	104.26.9.142	2020-06-02	2020-11-27
account.shodan.io	104.26.8.142	2020-06-02	2020-11-27
account.shodan.io	104.26.9.142	2020-01-25	2020-06-02
account.shodan.io	104.26.8.142	2020-01-25	2020-06-02
account.shodan.io	104.27.75.9	2019-05-25	2020-01-25
account.shodan.io	104.27.74.9	2019-05-25	2020-01-25
account.shodan.io	104.25.90.97	2017-07-16	2019-05-25
account.shodan.io	104.25.89.97	2017-07-16	2019-05-25
developer.shodan.io	172.67.75.154	2020-06-03	2020-11-27
developer.shodan.io	104.26.9.142	2020-06-03	2020-11-27
developer.shodan.io	104.26.8.142	2020-06-03	2020-11-27
developer.shodan.io	104.26.9.142	2020-01-24	2020-06-03
developer.shodan.io	104.26.8.142	2020-01-24	2020-06-03
developer.shodan.io	104.27.75.9	2019-05-30	2020-01-24
developer.shodan.io	104.27.74.9	2019-05-30	2020-01-24
developer.shodan.io	216.117.2.180	2018-04-05	2019-05-30
honeyscore.shodan.io	216.117.2.180	2019-08-26	2020-11-27
help.shodan.io	172.67.75.154	2020-06-02	2020-11-27
help.shodan.io	104.26.9.142	2020-06-02	2020-11-27
help.shodan.io	104.26.8.142	2020-06-02	2020-11-27
help.shodan.io	104.26.9.142	2020-01-25	2020-06-02
help.shodan.io	104.26.8.142	2020-01-25	2020-06-02
help.shodan.io	104.27.75.9	2019-05-27	2020-01-25
help.shodan.io	104.27.74.9	2019-05-27	2020-01-25
help.shodan.io	216.117.2.180	2018-12-16	2019-05-27
api.shodan.io	172.67.75.154	2020-08-06	2020-11-27
api.shodan.io	104.26.9.142	2020-08-06	2020-11-27

Table A.1: List of subdomains registered for shodan.io. This table ends on Page 84.

Subdomain	IP	First Seen	Last Seen
api.shodan.io	104.26.8.142	2020-08-06	2020-11-27
api.shodan.io	216.117.2.180	2018-04-05	2020-08-06
webcambrowser.shodan.io	216.117.2.180	2019-04-24	2020-11-27
shiptracker.shodan.io	45.55.163.215	2018-09-06	2020-11-27
images.shodan.io	172.67.75.154	2020-06-02	2020-11-27
images.shodan.io	104.26.9.142	2020-06-02	2020-11-27
images.shodan.io	104.26.8.142	2020-06-02	2020-11-27
images.shodan.io	104.26.9.142	2020-01-25	2020-06-02
images.shodan.io	104.26.8.142	2020-01-25	2020-06-02
images.shodan.io	104.27.75.9	2019-05-26	2020-01-25
images.shodan.io	104.27.74.9	2019-05-26	2020-01-25
images.shodan.io	104.25.90.97	2017-07-16	2019-05-26
images.shodan.io	104.25.89.97	2017-07-16	2019-05-26
malware-hunter.shodan.io	172.67.75.154	2020-10-24	2020-11-27
malware-hunter.shodan.io	104.26.9.142	2020-10-24	2020-11-27
malware-hunter.shodan.io	104.26.8.142	2020-10-24	2020-11-27
malware-hunter.shodan.io	216.117.2.180	2018-09-06	2020-10-24
ics-radar.shodan.io	216.117.2.180	2018-09-06	2020-11-27
icsmap.shodan.io	216.117.2.180	2018-09-06	2020-11-27
monitor.shodan.io	172.67.75.154	2020-06-03	2020-11-27
monitor.shodan.io	104.26.9.142	2020-06-03	2020-11-27
monitor.shodan.io	104.26.8.142	2020-06-03	2020-11-27
monitor.shodan.io	104.26.9.142	2020-01-25	2020-06-03
monitor.shodan.io	104.26.8.142	2020-01-25	2020-06-03
monitor.shodan.io	104.27.75.9	2019-07-20	2020-01-25
monitor.shodan.io	104.27.74.9	2019-07-20	2020-01-25
exploits.shodan.io	172.67.75.154	2020-06-03	2020-11-27
exploits.shodan.io	104.26.9.142	2020-06-03	2020-11-27
exploits.shodan.io	104.26.8.142	2020-06-03	2020-11-27
exploits.shodan.io	104.26.9.142	2020-01-25	2020-06-03
exploits.shodan.io	104.26.8.142	2020-01-25	2020-06-03
exploits.shodan.io	104.27.75.9	2019-05-26	2020-01-25
exploits.shodan.io	104.27.74.9	2019-05-26	2020-01-25
exploits.shodan.io	104.25.90.97	2018-06-15	2019-05-26
exploits.shodan.io	104.25.89.97	2018-06-15	2019-05-26

Subdomain	IP	First Seen	Last Seen
exploits.shodan.io	216.117.2.180	2018-04-05	2018-06-15
careers.shodan.io	216.117.2.180	2018-11-13	2020-11-27
dev.shodan.io	216.117.2.180	2019-07-28	2020-11-27
dojo.census.shodan.io	80.82.77.139	2018-09-05	2020-11-27
malware-hunter.census.shodan.io	66.240.205.34	2018-09-05	2020-11-27
nninja.census.shodan.io	216.117.2.180	2020-01-29	2020-11-27
census4.shodan.io	198.20.99.130	2018-04-05	2020-11-27
mqttmon.shodan.io	216.117.2.180	2020-05-30	2020-11-27
mqttmon.shodan.io	104.156.250.217	2018-10-27	2020-05-30
ncensus8.shodan.io	216.117.2.180	2020-01-29	2020-11-27
ncensus7.shodan.io	216.117.2.180	2020-01-28	2020-11-27
npirate.census.shodan.io	216.117.2.180	2020-01-29	2020-11-27
census9.shodan.io	71.6.167.142	2018-04-05	2020-11-27
tesla.census.shodan.io	71.6.147.254	2019-05-31	2020-11-27
imap.shodan.io	217.70.178.4	2018-03-11	2020-11-27
imap.shodan.io	217.70.184.9	2017-07-16	2018-03-11
maltego.shodan.io	216.117.2.180	2018-10-27	2020-11-27
refrigerator.census.shodan.io	71.6.146.130	2018-09-06	2020-11-27
2fwww.shodan.io	216.117.2.180	2019-11-26	2020-11-27
ns2.shodan.io	138.68.199.36	2017-07-16	2020-11-27
einstein.census.shodan.io	71.6.199.23	2019-06-07	2020-11-27
nwww.shodan.io	216.117.2.180	2019-07-23	2020-11-28
census1.shodan.io	198.20.69.74	2018-04-05	2020-11-28
wire.shodan.io	172.67.75.154	2020-06-02	2020-11-28
wire.shodan.io	104.26.9.142	2020-06-02	2020-11-28
wire.shodan.io	104.26.8.142	2020-06-02	2020-11-28
wire.shodan.io	104.26.9.142	2020-01-25	2020-06-02
wire.shodan.io	104.26.8.142	2020-01-25	2020-06-02
wire.shodan.io	104.27.75.9	2019-05-26	2020-01-25
wire.shodan.io	104.27.74.9	2019-05-26	2020-01-25
wire.shodan.io	104.25.90.97	2018-11-22	2019-05-26
wire.shodan.io	104.25.89.97	2018-11-22	2019-05-26
m247.ro.shodan.io	216.117.2.180	2020-02-27	2020-11-28
goldfish.census.shodan.io	216.117.2.180	2018-09-05	2020-11-28
nrefrigerator.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28

Subdomain	IP	First Seen	Last Seen
census.shodan.io	216.117.2.180	2018-04-05	2020-11-28
2000.shodan.io	172.67.75.154	2020-10-25	2020-11-28
2000.shodan.io	104.26.9.142	2020-10-25	2020-11-28
2000.shodan.io	104.26.8.142	2020-10-25	2020-11-28
2000.shodan.io	216.117.2.180	2019-04-01	2020-10-25
ndojo.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
nflower.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
ncensus12.shodan.io	216.117.2.180	2020-01-29	2020-11-28
census7.shodan.io	71.6.135.131	2018-04-05	2020-11-28
flower.census.shodan.io	94.102.49.190	2018-09-05	2020-11-28
snippets.shodan.io	172.67.75.154	2020-06-03	2020-11-28
snippets.shodan.io	104.26.9.142	2020-06-03	2020-11-28
snippets.shodan.io	104.26.8.142	2020-06-03	2020-11-28
snippets.shodan.io	104.26.9.142	2020-01-25	2020-06-03
snippets.shodan.io	104.26.8.142	2020-01-25	2020-06-03
snippets.shodan.io	104.27.75.9	2019-05-26	2020-01-25
snippets.shodan.io	104.27.74.9	2019-05-26	2020-01-25
snippets.shodan.io	104.25.90.97	2018-11-22	2019-05-26
snippets.shodan.io	104.25.89.97	2018-11-22	2019-05-26
ntesla.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
nhouse.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
cli.shodan.io	216.117.2.180	2018-10-27	2020-11-28
ninja.census.shodan.io	71.6.158.166	2018-09-05	2020-11-28
stream.shodan.io	45.77.97.111	2017-11-06	2020-11-28
stream.shodan.io	45.63.11.4	2017-11-06	2020-11-28
stream.shodan.io	209.222.10.250	2017-11-06	2020-11-28
stream.shodan.io	104.25.90.97	2017-08-20	2017-11-06
stream.shodan.io	104.25.89.97	2017-08-20	2017-11-06
stream.shodan.io	45.55.206.46	2017-07-16	2017-08-20
stream.shodan.io	162.243.186.86	2017-07-16	2017-08-20
stream.shodan.io	104.236.97.94	2017-07-16	2017-08-20
stream.shodan.io	104.236.227.23	2017-07-16	2017-08-20
stream.shodan.io	104.236.216.229	2017-07-16	2017-08-20
stream.shodan.io	104.236.209.12	2017-07-16	2017-08-20
nborder.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28

Subdomain	IP	First Seen	Last Seen
mason.census.shodan.io	89.248.167.131	2018-09-05	2020-11-28
wine.census.shodan.io	185.142.236.35	2020-07-28	2020-11-28
wine.census.shodan.io	216.117.2.180	2019-03-30	2020-07-28
webmail.shodan.io	216.117.2.180	2019-04-08	2020-11-28
webmail.shodan.io	217.70.178.6	2018-02-27	2019-04-08
webmail.shodan.io	217.70.182.5	2017-09-19	2018-02-27
webmail.shodan.io	192.168.23.1	2017-09-18	2017-09-19
webmail.shodan.io	217.70.182.5	2017-07-16	2017-09-18
census6.shodan.io	66.240.236.119	2018-07-31	2020-11-28
census6.shodan.io	37.1.211.129	2018-07-30	2018-07-31
census6.shodan.io	66.240.236.119	2018-04-05	2018-07-30
rancher.packet.dev.shodan.io	216.117.2.180	2019-01-03	2020-11-28
atlantic.census.shodan.io	188.138.9.50	2020-02-28	2020-11-28
burger.census.shodan.io	66.240.219.146	2018-09-05	2020-11-28
heimdal.metrics.shodan.io	45.63.21.128	2018-10-27	2020-11-28
census10.shodan.io	82.221.105.6	2018-04-05	2020-11-28
2fmonitor.shodan.io	216.117.2.180	2019-07-20	2020-11-28
inspire.census.shodan.io	216.117.2.180	2018-09-05	2020-11-28
ncensus9.shodan.io	216.117.2.180	2020-01-28	2020-11-28
census3.shodan.io	198.20.70.114	2018-04-05	2020-11-28
rancher.dev.shodan.io	159.89.178.191	2019-01-02	2020-11-28
board.census.shodan.io	71.6.147.198	2018-09-06	2020-11-28
www.shodan.io	172.67.75.154	2020-06-03	2020-11-28
www.shodan.io	104.26.9.142	2020-06-03	2020-11-28
www.shodan.io	104.26.8.142	2020-06-03	2020-11-28
www.shodan.io	104.26.9.142	2020-01-25	2020-06-03
www.shodan.io	104.26.8.142	2020-01-25	2020-06-03
www.shodan.io	104.27.75.9	2019-10-25	2020-01-25
www.shodan.io	104.27.74.9	2019-10-25	2020-01-25
www.shodan.io	104.25.90.97	2017-05-26	2019-10-25
www.shodan.io	104.25.89.97	2017-05-26	2019-10-25
ncensus6.shodan.io	216.117.2.180	2020-01-29	2020-11-28
census2.shodan.io	198.20.69.98	2018-04-05	2020-11-28
direct.shodan.io	216.117.2.180	2018-04-05	2020-11-28
ninspire.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28

Subdomain	IP	First Seen	Last Seen
9.shodan.io	216.117.2.180	2020-02-27	2020-11-28
status.shodan.io	172.67.75.154	2020-06-03	2020-11-28
status.shodan.io	104.26.9.142	2020-06-03	2020-11-28
status.shodan.io	104.26.8.142	2020-06-03	2020-11-28
status.shodan.io	104.26.9.142	2020-01-24	2020-06-03
status.shodan.io	104.26.8.142	2020-01-24	2020-06-03
status.shodan.io	104.27.75.9	2019-05-26	2020-01-24
status.shodan.io	104.27.74.9	2019-05-26	2020-01-24
status.shodan.io	104.25.90.97	2017-07-16	2019-05-26
status.shodan.io	104.25.89.97	2017-07-16	2019-05-26
hat.census.shodan.io	185.142.236.34	2019-12-05	2020-11-28
hat.census.shodan.io	216.117.2.180	2019-03-30	2019-12-05
cloud.census.shodan.io	94.102.49.193	2018-09-05	2020-11-28
house.census.shodan.io	89.248.172.16	2018-09-05	2020-11-28
sea.support.internal.shodan.io	142.93.28.101	2019-05-15	2020-11-28
nburger.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
smtp.shodan.io	217.70.178.3	2018-03-11	2020-11-28
smtp.shodan.io	217.70.184.8	2017-07-16	2018-03-11
battery.census.shodan.io	93.174.95.106	2018-09-05	2020-11-28
ncensus10.shodan.io	216.117.2.180	2020-01-29	2020-11-28
nsky.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
ncensus11.shodan.io	216.117.2.180	2020-01-29	2020-11-28
vita.census.shodan.io	216.117.2.180	2019-03-12	2020-11-28
census12.shodan.io	71.6.165.200	2018-04-05	2020-11-28
pop.shodan.io	217.70.178.4	2018-03-11	2020-11-28
pop.shodan.io	217.70.184.9	2017-07-16	2018-03-11
nmalware-hunter.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
nhat.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
ics-apac-2017.shodan.io	216.117.2.180	2019-12-01	2020-11-28
turtle.census.shodan.io	216.117.2.180	2018-09-05	2020-11-28
ncensus3.shodan.io	216.117.2.180	2020-01-29	2020-11-28
pacific.census.shodan.io	85.25.103.50	2018-09-05	2020-11-28
nmason.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
neinstein.census.shodan.io	216.117.2.180	2020-01-29	2020-11-28
blog.shodan.io	172.67.75.154	2020-06-03	2020-11-28

Subdomain	IP	First Seen	Last Seen
blog.shodan.io	104.26.9.142	2020-06-03	2020-11-28
blog.shodan.io	104.26.8.142	2020-06-03	2020-11-28
blog.shodan.io	104.26.9.142	2020-01-24	2020-06-03
blog.shodan.io	104.26.8.142	2020-01-24	2020-06-03
blog.shodan.io	104.27.75.9	2019-05-25	2020-01-24
blog.shodan.io	104.27.74.9	2019-05-25	2020-01-24
blog.shodan.io	104.25.90.97	2018-10-20	2019-05-25
blog.shodan.io	104.25.89.97	2018-10-20	2019-05-25
blog.shodan.io	104.236.198.48	2017-07-16	2018-10-20
dlink-report.shodan.io	216.117.2.180	2018-09-06	2020-11-28
enterprise.shodan.io	172.67.75.154	2020-06-02	2020-11-28
enterprise.shodan.io	104.26.9.142	2020-06-02	2020-11-28
enterprise.shodan.io	104.26.8.142	2020-06-02	2020-11-28
enterprise.shodan.io	104.26.9.142	2020-01-25	2020-06-02
enterprise.shodan.io	104.26.8.142	2020-01-25	2020-06-02
enterprise.shodan.io	104.27.75.9	2019-05-26	2020-01-25
enterprise.shodan.io	104.27.74.9	2019-05-26	2020-01-25
enterprise.shodan.io	104.25.90.97	2018-10-27	2019-05-26
enterprise.shodan.io	104.25.89.97	2018-10-27	2019-05-26
static.shodan.io	172.67.75.154	2020-06-02	2020-11-28
static.shodan.io	104.26.9.142	2020-06-02	2020-11-28
static.shodan.io	104.26.8.142	2020-06-02	2020-11-28
static.shodan.io	104.26.9.142	2020-01-25	2020-06-02
static.shodan.io	104.26.8.142	2020-01-25	2020-06-02
static.shodan.io	104.27.75.9	2019-05-26	2020-01-25
static.shodan.io	104.27.74.9	2019-05-26	2020-01-25
static.shodan.io	104.25.90.97	2019-02-15	2019-05-26
static.shodan.io	104.25.89.97	2019-02-15	2019-05-26
static.shodan.io	37.1.211.129	2018-07-24	2019-02-15
static.shodan.io	104.25.90.97	2017-07-16	2018-07-24
static.shodan.io	104.25.89.97	2017-07-16	2018-07-24
ns1.shodan.io	162.243.164.140	2017-07-16	2020-11-28
nwine.census.shodan.io	216.117.2.180	2020-01-30	2020-11-28
nblog.shodan.io	216.117.2.180	2019-07-24	2020-11-28
maptiles.shodan.io	172.67.75.154	2020-06-03	2020-11-28

A.3 Virtual Machine Setup Script

The below script was used to configure our VMs during our empirical study in Section 3.1.1 on Page 36.

```
1  #!/bin/bash
2  ssh ec2-user@<IP> -i key.pem
3  yum update
4  yum upgrade
5  yum install tcpdump httpd vsftpd postfix
6  vim /etc/httpd/conf/httpd.conf
7  #add Listen 443 under Listen 80
8  vim /etc/postfix/main.cf
9  #Uncomment interfaces = ALL
10 service httpd start
11 service httpd start
12 service vsftpd start
13 service postfix start
14 tcpdump -i any -G 86400 -K -n -w %B%d.pcap &
```

Bibliography

[1] Cartopy 0.18.0. Available at https://pypi.org/project/Cartopy/.

[2] Censys About. Available at https://censys.io/company. Accessed Oct 2020.

[3] Censys Search. Available at https://censys.io/ipv4.

[4] Internet Assigned Numbers Authority. https://www.iana.org/.

[5] National Vulnerability Database. National Vulnerability Database. https://nvd.nist.gov/.

[6] pcap-splitter 1.0.0. Available at https://pypi.org/project/pcap-splitter/.

[7] SecuirtyTrails - The Total Internet Inventory. https://securitytrails.com/.

[8] Tor: Onion Service Protocol. https://2019.www.torproject.org/docs/onion-services.

[9] Tor Project. https://www.torproject.org/.

[10] What Is Shodan? Available at https://help.shodan.io/the-basics/what-is-shodan. Accessed Oct 2020.

[11] WhoisXMLAPI. https://ip-geolocation.whoisxmlapi.com/.

[12] The ZMap Project. https://zmap.io/.

[13] Zoomeye. https://www.zoomeye.org/.

[14] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17, 2015.

[15] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *USENIX Security Symposium*, pages 1093–1110, 2017.

[16] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. STD 66, RFC Editor, January 2005. http://www.rfc-editor.org/rfc/rfc3986.txt.

[17] Tim Berners-Lee, Roy T. Fielding, and Henrik Frystyk Nielsen. Hypertext Transfer Protocol HTTP/1.0. RFC 1945, RFC Editor, May 1996. `http://www.rfc-editor.org/rfc/rfc1945.txt`.

[18] Hugo LJ Bijmans, Tim M Booij, and Christian Doerr. Just the Tip of the Iceberg: Internet-Scale Exploitation of Routers for Cryptojacking. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 449–464, 2019.

[19] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 3546, RFC Editor, June 2003.

[20] Roland Bodenheim, Jonathan Butts, Stephen Dunlap, and Barry Mullins. Evaluation of the Ability of the Shodan Search Engine to Identify Internet-Facing Industrial Control Devices. *International Journal of Critical Infrastructure Protection*, 7(2):114–123, 2014.

[21] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. Directed Greybox Fuzzing. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 2329–2344, 2017.

[22] S. Bradner and V. Paxson. Iana Allocation Guidelines for Values in the Internet Protocol and Related Headers. BCP 37, RFC Editor, March 2000.

[23] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. 1998.

[24] D. Carlin, J. Burgess, P. O'Kane, and S. Sezer. You Could Be Mine(d): The Rise of Cryptojacking. *IEEE Security & Privacy*, 18(2):16–22, 2020.

[25] Yongle Chen, Xiaowei Lian, Dan Yu, Shichao Lv, Shaochen Hao, and Yao Ma. Exploring Shodan From the Perspective of Industrial Control Systems. 8:75359–75369, 2020.

[26] Michael Chertoff. A Public Policy Perspective of the Dark Web. *Journal of Cyber Policy*, 2(1):26–38, 2017.

[27] Roger Christopher. *Port Scanning Techniques and the Defense Against Them*, 2001. Available at `https://www.sans.org/reading-room/whitepapers/auditing/paper/70`. Accessed July 2020.

[28] Catalin Cimpanu. Ichidan. Available at `https://www.bleepingcomputer.com/news/security/ichidan-is-a-shodan-like-search-engine-for-the-dark-web/`, Accessed Nov 2020.

[29] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. BCP 165, RFC Editor, August 2011.

[30] W. Bruce Croft, Donald Metzler, and Trevor Strohman. Search Engines - Information Retrieval in Practice. 2009.

[31] Jakub Dalek, Bennett Haselton, Helmi Noman, Adam Senft, Masashi Crete-Nishihata, Phillipa Gill, and Ronald J Deibert. A Method for Identifying and Confirming the Use of URL Filtering Products for Censorship. In *Internet Measurement Conference*, pages 23–30, 2013.

[32] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. STD 86, RFC Editor, July 2017.

[33] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor, August 2008. `http://www.rfc-editor.org/rfc/rfc5246.txt`.

[34] Cristian Duda, Gianni Frey, Donald Kossmann, and Chong Zhou. Ajaxsearch: Crawling, Indexing and Searching Web 2.0 Applications. *Proceedings of the VLDB Endowment*, 1(2):1440–1443, 2008.

[35] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. A Search Engine Backed by Internet-wide Scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[36] Zakir Durumeric, Michael Bailey, and J Alex Halderman. An Internet-wide View of Internet-wide Scanning. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 65–78, 2014.

[37] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. ZMap: Fast Internet-wide Scanning and its Security Applications. In *Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, 2013.

[38] Vincent J Ercolani, Mark W Patton, and Hsinchun Chen. Shodan Visualized. In *Conference on Intelligence and Security Informatics (ISI)*, pages 193–195. IEEE, 2016.

[39] Brown Farinholt, Mohammad Rezaeirad, Paul Pearce, Hitesh Dharmdasani, Haikuo Yin, Stevens Le Blond, Damon McCoy, and Kirill Levchenko. To Catch a Ratter: Monitoring the Behavior of Amateur Darkcomet Rat Operators in the Wild. In *IEEE Symposium on Security and Privacy (SP)*, pages 770–787, 2017.

[40] Kristin M Finklea. Dark Web, 2015. `https://a51.nl/sites/default/files/pdf/R44101%20(1).pdf`, Accessed Nov 2020.

[41] Béla Genge and Călin Enăchescu. ShoVAT: Shodan-based Vulnerability Assessment Tool for Internet-facing Services. *Security and communication networks*, 9(15):2696–2714, 2016.

[42] F. Gont and T. Chown. Network Reconnaissance in IPv6 Networks. RFC 7707, RFC Editor, March 2016.

[43] Hwanjo Heo and Seungwon Shin. Who is Knocking on the Telnet Port: A Large-Scale Empirical Study of Network Scanning. In *Asia Conference on Computer and Communications Security*, page 625–636, New York, NY, USA, 2018.

[44] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web. *World Wide Web Consortium (W3C)*, 2004.

[45] Doowon Kim, Bum Jun Kwon, and Tudor Dumitraş. Certified malware: Measuring Breaches of Trust in the Windows Code-Signing PKI. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1435–1448, 2017.

[46] M. Komu, M. Bagnulo, K. Slavov, and S. Sugimoto. Sockets Application Program Interface (API) for Multihoming Shim. RFC 6316, RFC Editor, July 2011.

[47] M. Koster. A Method for Web Robots Control. Technical report, INTERNET DRAFT.

[48] Platon Kotzias, Leyla Bilge, Pierre-Antoine Vervier, and Juan Caballero. Mind Your Own Business: A Longitudinal Study of Threats and Vulnerabilities in Enterprises. In *NDSS*, 2019.

[49] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. "I Have No Idea What I'm Doing"-On the Usability of Deploying HTTPS. In *26th USENIX Security Symposium Security 17)*, pages 1339–1356, 2017.

[50] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. All Things Considered: an Analysis of IoT Devices on Home Networks. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1169–1185, 2019.

[51] Deepak Kumar, Zhengping Wang, Matthew Hyder, Joseph Dickinson, Gabrielle Beck, David Adrian, Joshua Mason, Zakir Durumeric, J Alex Halderman, and Michael Bailey. Tracking Certificate Misissuance in the Wild. In *IEEE Symposium on Security and Privacy (SP)*, pages 785–798. IEEE, 2018.

[52] Gaëtan Leurent and Thomas Peyrin. SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1839–1856, 2020.

[53] Frank Li, Zakir Durumeric, Jakub Czyz, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson. You've Got Vulnerability: Exploring Effective Vulnerability Notifications. In *USENIX Security Symposium*, pages 1033–1050, 2016.

[54] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's Deep Web Crawl. *Proceedings of the VLDB Endowment*, 1(2):1241–1252, 2008.

[55] John Matherly. *Complete Guide to Shodan*. Learnpub, 2015.

[56] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J Alex Halderman, et al. An Internet-wide View of ICS Devices. In *Conference on Privacy, Security and Trust (PST)*, pages 96–103. IEEE, 2016.

[57] Yisroel Mirsky, Tom Mahler, Ilan Shelef, and Yuval Elovici. CT-GAN: Malicious Tampering of 3D Medical Imagery Using Deep Learning. In *USENIX Security Symposium*, pages 461–478, 2019.

[58] Debajyoti Mukhopadhyay, Aritra Banik, Sreemoyee Mukherjee, Jhilik Bhattacharya, and Young-Chon Kim. A Domain Specific Ontology Based Semantic Web Search Engine. *arXiv*, 2011.

[59] Austin Murdock, Frank Li, Paul Bramsen, Zakir Durumeric, and Vern Paxson. Target Generation for Internet-wide IPv6 Scanning. In *Internet Measurement Conference*, pages 242–253, 2017.

[60] J. Postel. DoD Standard Internet Protocol. RFC 760, RFC Editor, January 1980.

[61] Jon Postel. Transmission Control Protocol. STD 7, RFC Editor, September 1981. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[62] Stephen Pritchard. Shodan Founder John Matherly on IoT Security, Dual-Purpose Hacking Tools, and Information Overload. Available at `https://portswigger.net/daily-swig/shodan-founder-john-matherly-on-iot-security-dual-purpose-hacking-tools-and-information-overload`. Accessed Oct 2020.

[63] Ram Sundara Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. Measuring the Deployment of Network Censorship Filters at Global Scale. In *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[64] J. Reynolds and J. Postel. Assigned Numbers. RFC 1340, RFC Editor, July 1992.

[65] Mohammad Rezaeirad, Brown Farinholt, Hitesh Dharmdasani, Paul Pearce, Kirill Levchenko, and Damon McCoy. Schrödinger's RAT: Profiling the Stakeholders in the Remote Access Trojan Ecosystem. In *USENIX Security Symposium*, pages 1043–1060, 2018.

[66] Philipp Richter and Arthur Berger. Scanning the Scanners: Sensing the Internet from a Massively Distributed Network Telescope. In *Proceedings of the Internet Measurement Conference*, pages 144–157, 2019.

[67] Dakota S Rudesill, James Caverlee, and Daniel Sui. The Deep Web and the Darknet: A Look Inside the Internet's Massive Black Box. *Woodrow Wilson International Center for Scholars, STIP*, 3, 2015.

[68] Tom Seymour, Dean Frantsvog, and Satheesh Kumar. History of Search Engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4):47–58, 2011.

[69] Drew Springall, Zakir Durumeric, and J Alex Halderman. FTP: The Forgotten Cloud. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 503–513. IEEE, 2016.

[70] Drew Springall, Zakir Durumeric, and J Alex Halderman. Measuring the Security Harm of TLS Crypto Shortcuts. In *Internet Measurement Conference*, pages 33–47, 2016.

[71] Benjamin VanderSloot, Johanna Amann, Matthew Bernhard, Zakir Durumeric, Michael Bailey, and J Alex Halderman. Towards a Complete View of the Certificate Ecosystem. In *Internet Measurement Conference*, pages 543–549, 2016.

[72] Gerry Wan, Liz Izhikevich, David Adrian, Katsunari Yoshioka, Ralph Holz, Christian Rossow, and Zakir Durumeric. On the Origin of Scanning: The Impact of Location on Internet-wide Scans. In *ACM Internet Measurement Conference*, October 2020.

[73] Xueqiang Wang, Yuqiong Sun, Susanta Nanda, and XiaoFeng Wang. Looking from the Mirror: Evaluating IoT Device Security Through Mobile Companion Apps. In *USENIX Security Symposium*, pages 1151–1167, 2019.

[74] Gabriel Weimann. Going Dark: Terrorism on the Dark Web. *Studies in Conflict & Terrorism*, 39(3):195–206, 2016.