

A Survey of Shodan Data

By

Vincent Ercolani

A Master's Paper Submitted to the Faculty of the

DEPARTMENT OF MANAGEMENT INFORMATION SYSTEMS

ELLER COLLEGE OF MANAGEMENT

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2017

STATEMENT BY AUTHOR

This paper has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona.

Brief quotations from this paper are allowable without special permission, provided that an accurate acknowledgement of the source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part must be obtained from the author.

SIGNED: Vincent Ercolani

APPROVAL BY MASTERS PAPER ADVISOR

This paper has been approved on the date shown below:

Dr. Mark Patton

Lecturer of Management Information Systems

05/05/2017

Date

Table of Contents

STATEMENT BY AUTHOR	i
Table of Contents	ii
List of Figures	vi
List of Tables	vii
1 INTRODUCTION	1
2 BACKGROUND/LITERATURE REVIEW	1
2.1 Literature Review	1
2.1.1 Cyber Threat Intelligence (CTI).....	1
2.1.2 Shodan.....	1
2.1.3 Security Visualizations.....	2
2.1.4 Research Gaps	3
3 METHODOLOGY.....	4
3.1 Introduction	4
3.2 Approach	4
4 DATA COLLECTION	5
4.1 Introduction	5
4.1.1 Shodan Dumps	5
4.1.2 Data Processing	5
4.1.3 Data Store.....	7
4.2 Tool Creation	7

5 Data Processing	8
5.1 Introduction	8
5.2 ICS Identification	9
5.3 Data Sampling	11
5.4 Feature Selection & Analysis	11
5.4.1 Reverse DNS & IP Locations	12
5.4.2 Change in Scans over time	13
6 Data Visualization	14
6.1 Changes Over Time	14
6.2 Finding Malicious Protocols	17
6 Conclusion	19
APPENDIX A – Shodan Data	20
Main Properties	20
Opt Properties	22
SSL Properties	23
APPENDIX B – Shodan Tables	24
Information Tables	25
ShodanModules	25
Core Data	26
IPv4_Scans_WW	27
IPAddresses	27
Locations	28
Banner Data	28

Banners_WW	29
Scan_Banner_WW.....	29
Banner_Dates_WW.....	29
Common Properties	29
Common_Properties_WW	30
Host / Domain Data	30
Host_Dom_Properties_WW.....	30
HTML Data.....	30
HTML_WW	31
Scan_HTML_WW.....	31
IP Address History.....	31
IPAddress_History_WW.....	31
Opt Data.....	32
Opt_Properties_WW.....	32
Vulnerability Opt Data	32
Opt_Prop_Vuln_WW.....	32
SSL Data	33
Scan_SSL_WW.....	33
SSL_Properties_WW.....	33
Properties.....	33
Properties_WW.....	33
Appendix C – Shodan Database ERD	34
Appendix D – Shodan Modules and Classifications	35

Appendix D – Shodan Modules and Classifications	43
parseShodanFile.py.....	43
shodan_sql_import.py	54
REFERENCES	60

List of Figures

Figure 1: Research Design	4
Figure 2: Section bitmask values	8
Figure 3: Venezuela ICS devices Aug 2016 [Afarin, 2017]	15
Figure 4: Venezuela ICS devices Jan 2017 [Afarin, 2017]	16
Figure 5: Arizona PLC5 IPs by Modularity	17
Figure 6: Arizona plc5 IPs by protocol type	18
Figure 7: Shodan Database ERD	34

List of Tables

Table 1: Shodan IPv4 Scans per Week.....	7
Table 2: Shodan parser input parameters.....	7
Table 3: Shodan modules communicating with IOT devices	9
Table 4: Shodan modules communicating with ICS devices.....	10
Table 5: Top 20 ICS scans per state.....	11
Table 6: Scans where country code of IP and domain differ	12
Table 7: Change in frequency of scans	13
Table 8: Main Shodan Object Properties	21
Table 9: Optional Shodan Object Properties	22
Table 10: SSL Object Properties in Shodan Object.....	23
Table 11: Parser and Data store Sections.....	24
Table 12: Shodan Modules	25
Table 13: IPv4_Scans_WW	27
Table 14: IPAddresses	28
Table 15: Locations.....	28
Table 16: Banners_WW.....	29
Table 17: Scan_Banner_WW	29

Table 18: Banner_Dates_WW	29
Table 19: Common_Properties_WW	30
Table 20: Host_Dom_Properties_WW	30
Table 21: HTML_WW	31
Table 22: Scan_HTML_WW	31
Table 23: IPAddress_History_WW	31
Table 24: Opt_Properties_WW	32
Table 25: Opt_Prop_Vuln_WW	32
Table 26: Scan_SSL_WW	33
Table 27: SSL_Properties_WW	33
Table 28: Properties_WW	33
Table 29: Shodan Modules	42

1 INTRODUCTION

This is a survey of information obtainable from the Shodan internet database with a discussion of the limitations.

The devices on the internet continuously expands as many non-traditional items are introduced and connected. Items ranging from personal to business devices are being rapidly introduced and many of them are now being connected to the internet and are visible to online searches, yet security often seems to take a back seat to features, functionality, and connectivity despite growing cyber threats to all online devices. In 2016 there were 189 new vulnerabilities in Industrial Control Systems (ICS) components published, most of them deemed to have medium (42%) or critical (49%) severity levels (Andreeva, 2016). Because the industrial sector is critically important to a smoothly functioning society and quality of life, threats to this sector are considered significant and as such it is key area of interest for cybersecurity research.

The search engine Shodan is one tool often utilized to better understand the landscape for industrial control systems. Shodan is a database connected to an internet portal that leverages ongoing scans of the internet to find open ports on the IPv4 address space. From the Shodan data, one can search for and analyze services, ports, IP addresses, and other relevant information to industrial control systems (ICS) and/or supervisory control and data acquisition devices (SCADA).

Shodan captures many attributes for each IP/port scanned, and adds additional attributes based on the original scan data. Understanding these fields, their reliability, limitations, and meaning is critical to conducting research and analysis which can lead to insights for cyber threat intelligence (CTI) and related cybersecurity initiatives. Cyber threat intelligence needs to be relevant and accurate; therefore Shodan data must be evaluated for consistency and the stability of attributes

reported over time. This research implements security visualizations (VizSec) to survey Shodan data across a specified dataset over time, allowing us to understand the identification process of ICS/SCADA devices and evaluate Shodan's data and data attributes for completeness with respect to CTI. The motivation for this research stems from the need to understand Shodan data with respect to ICS/SCADA, understand the limits and variability of internet scan results over time, and ultimately determine if Shodan can be used as a reliable data source for CTI information specifically in the ICS/ SCADA domains.

2 BACKGROUND/LITERATURE REVIEW

2.1 Literature Review

To form the basis of this research, we reviewed three major areas of literature to include: Cyber Threat Intelligence, Shodan, and Security Visualizations. The following sections discuss the findings within each area.

2.1.1 Cyber Threat Intelligence (CTI)

Many organizations rely on Cyber Threat Intelligence (CTI) data and reports to mitigate cyber threats. CTI is threat intelligence related to computers and networks (Shackleford, 2015). It is important to understand that CTI focuses on mitigating potential or pending attacks before they occur (EY, 2014). It should be noted that one piece of literature was unable to find any CTI portals leveraging machine learning or data mining techniques to gain a deeper understanding of their data (Samanti, 2016). Therefore analyzing the Shodan data through data mining techniques for the purpose of providing CTI does hold value.

2.1.2 Shodan

Shodan, a search engine for the Internet of Things (IoT) can help provide a novel data source for CTI visualizations (Bodenheim, 2014). Through a Cyber Threat Intelligence course at the University of Arizona we know Shodan does the following:

- Crawls, scans, and indexes billions of devices on the Internet
- Uses various search filters so users can find indexed devices including Supervisory Control and Data Acquisition (SCADA) devices (Bodenheim, 2014).
- Returns network (port, IP), location (latitude, longitude), and device specific (banner, protocol) data pertaining to the devices (Bodenheim, 2014).

- Prior Shodan explorations have used specific keywords, either manually through the web interface (Bodenheim et al., 2014) or automatically through the API (Patton et al., 2014) to identify SCADA devices. (Samtani, 2016).

Through Shodan we can understand device exposure amongst the internet. It should be noted that previous work on the Shodan database has shown devices are generally indexed within 3 weeks of coming online (Ercolani, 2016). Additional work done on evaluating Shodan as a scanning tool does show inconsistency in results from scans. However, analysis of the data from the scans has not been delved into for deeper analysis. Findings from Rohrmann's research show discrepancies from his own scans to that of Shodan over the same IP range over the same time frame (Rohrmann, 2017).

2.1.3 Security Visualizations

Information visualization is one of these tools that allows users to understand the behavior of the managed network (Guimaraes, 2015). Visualization alternatives include: 3D, network graph, line/bar chart, radial methods, trees and geographical maps. (Langton, 2010; McKenna, 2015). Security metrics, security monitoring, anomaly detection, forensics, and malware analysis are examples where data visualization can play a vital role (SANS, 2014). Even with the information that many studies identify accessible and vulnerable SCADA systems (Arnaert, 2015), a lack of literature exists in utilizing visualization techniques with the type of data from Shodan. Through a case study performed by Cyrus Afarin, security visualization is performed on a small scaled focused group. The techniques utilized are for representing the changes (if any) from Shodan data over time through security visualizations (Afarin, 2017).

2.1.4 Research Gaps

Based on the literature review of cyber threat intelligence, Shodan, and security visualizations, the following gaps were identified:

- Visualization techniques are not being used for device identification on the internet
- Cyber Threat Intelligence does not take advantage of data mining techniques or proactive visualizations using Shodan or comparable data sets to enhance data comprehension and hence cyber posture
- Shodan scans have been evaluated to identify if differences exist through self-conducted scans (Rohrmann, 2017) and over time evaluations (Afarin, 2017) but specific data variances have not been systematically analyzed
- Larger scaled visualizations over an increased dataset over time are necessary due to the evolving nature of ICS/SCADA devices coming online as well as the growing cyber threats to internet enabled devices. The visualizations clarify the risks and posture within networks and provide insights on where to focus efforts

Therefore, this research proposes to further efforts performed by Afarin and Rohrmann by performing a deep analysis of Shodan data gathered and visualized to better understand data variances and inconsistencies over time (Afarin, 2017; Rohrmann, 2017). Additionally, the research will attempt to address whether visualizations can be created that will be effective in identifying devices through the understanding of open ports and the protocols running on these ports (specifically for SCADA/ICS). Through the methods of visualizations, data analysis, trend identification, and active CTI development this work aims to evaluate Shodan and the value it's data can provide.

3 METHODOLOGY

3.1 Introduction

To get a better understanding of Shodan data we need unfettered access to the data that is collected. The most complete approach is to forgo the online API and work with complete daily dumps of Shodan scan data. Access was obtained to the daily gzip files of Shodan scans. These files range from 75 to over 100 gigabytes in size and contain upwards of 40 million rows a day. All records are kept in a JSON format that is only semi structured. This created several challenges that needed to be overcome in order to store and work with the data.

3.2 Approach

To address the research questions a design was created centered on three main steps: data collection, data processing, and visualization creation. Figure 1: Research Design gives an overview of the research design.

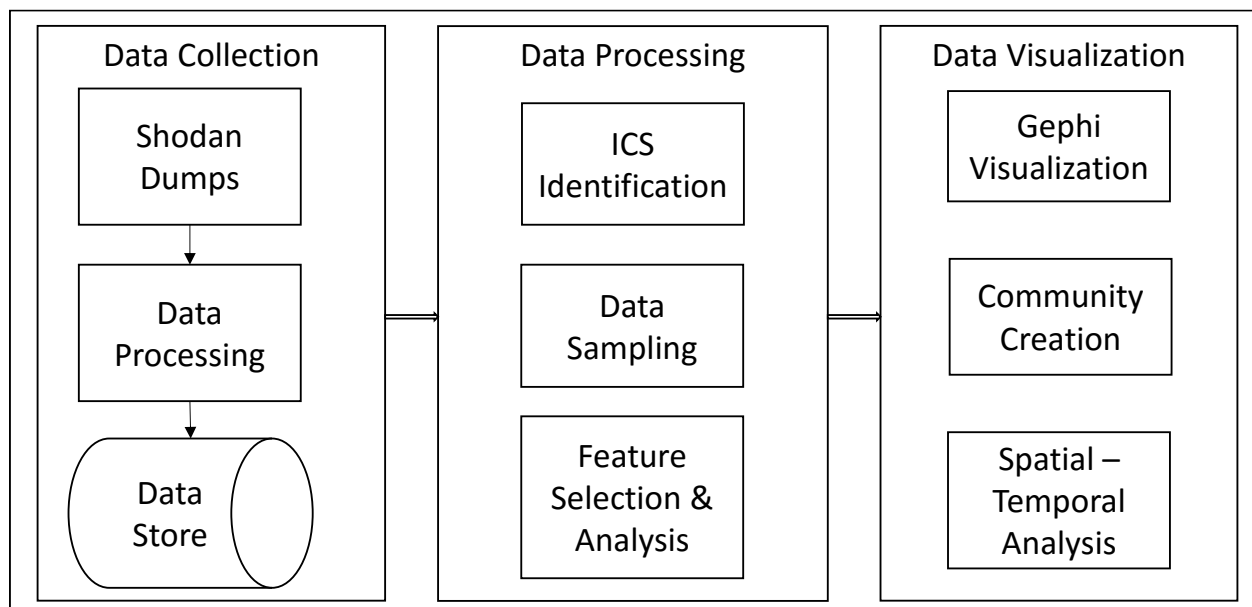


Figure 1: Research Design

4 DATA COLLECTION

4.1 Introduction

The initial step in the collection of data was obtaining and analyzing the daily data dumps downloaded from Shodan. These are gzipped files containing one JSON object per line representing all data that Shodan has gathered from a successful scan. The dumps were reverse engineered to understand the JSON objects that contained Shodan scans. Care was taken to insure that relevant portions of the object were in a format that could be easily put in a database and to insure that there was no loss of the unstructured data. Finally, a database was designed and a tool was developed to parse the files and load the data into the extensible database design.

4.1.1 Shodan Dumps

Since dumps contained 30 to 45 million rows per files a script was written to extract portions of the file for closer inspection. Shodan scan object were analyzed to determine the structure. Knowledge of the Shodan scan object structure can be explored in more depth in [Appendix A](#).

4.1.2 Data Processing

In the process of parsing the Shodan scan object data the information within the object was separated into ten distinct sections. These sections allowed the analysis of only the portions of the record object that would be of research interest. Files would be able to be processed and only the key information needed would be saved to a database. These sections included:

- Core Data
- Banner Data
- Common Properties
- Host / Domain Data

- HTML Data
- IP Address History
- Opt (Optional) Data
- Vulnerability Opt Data
- SSL Data
- Properties

Further explanation of the sections please refer to Table 11: Parser and Data store Sections in [Appendix B](#).

Care was taken to insure that when information in a Shodan scan object changed all information would be processed and stored. Crawler information (information identifying the specific Shodan crawling server which executed the recorded scan) is captured but it is stored in the properties table within the database. This makes it difficult to use with the current structure of the database.

One important piece of data that had proven invaluable within the Shodan scan object is the shodanModule field. This field gave the name of the specific Shodan module being employed by the crawling server, or crawler, to communicate on the port. This gives us the best indication of what protocol the port is running as Shodan has the ability to communicate using over 150 different protocols. These protocols were captured and classified to facilitate research on protocols of interest. Sixty-one of the Shodan modules still remain unclassified at this time. The main classifications of interest to this research are ICS classified protocols. A full list of current protocol classes are located in [Appendix B](#).

4.1.3 Data Store

Once a complete understanding of Shodan scan objects was gained and the data processing was worked out I created a design for the database that would be able to hold the parsed data. An entity relation diagram (ERD) of the database is in Appendix C – Shodan Database ERD, the related data dictionary is in APPENDIX B – Shodan Tables. These appendices hold all information needed to understand the database.

The database loads information according to ISO weeks. This shards the data and limits table sizes to facilitate analytics. Even with sharding the data the tables are still huge, the eight weeks of data that were used for this research are described in Table 1: Shodan IPv4 Scans per Week.

	Num Records
2016, Week 31	271,408,627
2016, Week 32	283,806,490
2016, Week 33	293,899,514
2016, Week 34	262,092,886
2017, Week 01	273,706,070
2017, Week 02	295,574,020
2017, Week 03	278,058,182
2017, Week 04	236,070,732

Table 1: Shodan IPv4 Scans per Week

This leads to almost 2.2 billion scans stored for further analysis over an 8 week period.

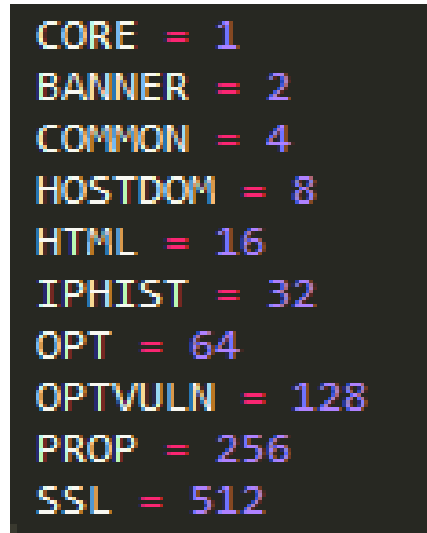
4.2 Tool Creation

A parser was created for parsing the daily dump files obtained from Shodan. This parser was written in python. The main executable is parseShodanFile.py, which required the python script shodan_sql_import.py to create the SQL insert scripts. The arguments to run the file are described in the following table.

Tag	Alt Tag	Type	Required	Default	Description
-g	--gzipfile	String	Yes		The gzip file to be parsed
-u	--update	Integer	No	10000	Print update to console every [x] number of lines in the file
-s	--sections	Integer	No	1023	Bitmask of the sections to be parsed from the file
-S	--skip	Integer	No	0	Number of lines to skip before parsing begins
-f	--flushfreq	Integer	No	8	Flush print buffers every [y] updates
-v	--verbose	True	No	False	Display verbose output of parsing actions

Table 2: Shodan parser input parameters

All parsed files are put into a directory called `[filename]_parsed`. Daily dump gzip files are moved to a completed directory when parsing is complete. Files in the parsed directory include the CSV (Comma Separated Variable) files for loading into each table as well as the SQL script for doing the loads of each section. Sections should be loaded into the database in the order of the sections in Figure 2: Section bitmask values. By adding the values of the section that you want parsed it is possible to only have those sections pulled out for inserting into the database. Additional sections can be parsed at a later time if a user would like to expand the information available.



```
CORE = 1
BANNER = 2
COMMON = 4
HOSTDOM = 8
HTML = 16
IPHIST = 32
OPT = 64
OPTVULN = 128
PROP = 256
SSL = 512
```

Figure 2: Section bitmask values

5 Data Processing

5.1 Introduction

Through experimentation, discussions with John Matherly (the creator of Shodan), and gaining a better understanding of the Shodan scan object I have found that additional data has been included that is not directly related to performing the scan. Many additional data gathering steps had already been performed at the time of the scan, including:

- Reverse DNS lookup: gets the name and domain for an IP
- Whois Lookup to get the ISP and Organization associated with the IP Address
- IP locations: gets the location of the IP
- Checks for vulnerabilities
 - o Check for susceptibility to heart bleed bug

- Tags for a scan
 - o ICS, IOT, VPN, Database, Honeygot, TOR, Medical

Beyond information that can be gathered from the JSON object of a scan these items can be put together to further CTI knowledge.

5.2 ICS Identification

Shodan has access to over 150 communication protocols used by devices on the web. Shodan created shodanModules to communicate using these protocols. Scans are saved based on finding a port open and a successful communication using one of these shodanModules. Using information gathered from the Shodan site and looking up shodanModule names, industrial control services (ICS) and internet of things (IOT) protocols were identified and classified. Table 3: Shodan modules communicating with IOT devices & Table 4: Shodan modules communicating with ICS devices display the Shodan modules for IOT and ICS devices respectively.

Shodan Module	Class	Description
dahua-dvr	IOT	Grab the serial number from a Dahua DVR device.
flux-led	IOT	Grab the current state from a Flux LED light bulb.
gardasoft-vision	IOT	Grabs the version for the Gardasoft controller.
hifly	IOT	Checks whether the HiFly lighting control is running.
idevice	IOT	Connects to an iDevice and grabs the property list.
ikettle	IOT	Check whether the device is a coffee machine/ kettle.
moxa-nport	IOT	Attempts to grab information from Moxna Nport devices.
mqtt	IOT	Grab a list of recent messages from an MQTT broker.
smarter-coffee	IOT	Checks the device status of smart coffee machines.
wemo-http	IOT	Connect to a Wemo Link and grab the setup.xml file
yahoo-smarttv	IOT	Checks whether the device is running the Yahoo Smart TV device communication service.

Table 3: Shodan modules communicating with IOT devices

Shodan Module	Class	Description
automated-tank-gauge	ICS	Get the tank inventory for a gasoline station.
bacnet	ICS	Gets various information from a BACnet device.
coap	ICS	Check whether the server supports the CoAP protocol
codesys	ICS	Grab a banner for Codesys daemons
dnp3	ICS	A dump of data from a DNP3 outstation
ethernetip	ICS	Grab information from a device supporting EtherNet/IP over TCP
ethernetip-udp	ICS	Grab information from a device supporting EtherNet/IP over UDP
fox	ICS	Grabs a banner for proprietary FOX protocol by Tridium
general-electric-srtp	ICS	Check whether the GE SRTP service is active on the device.
hart-ip-udp	ICS	Checks whether the IP is a HART-IP gateway.
iec-104	ICS	Banner grabber for the IEC-104 protocol.
kamstrup	ICS	Kamstrup Smart Meters
knx	ICS	Grabs the description from a KNX service.
lantronix-udp	ICS	Attempts to grab the setup object from a Lantronix device.
matrikon-opc	ICS	Checks whether the device is running Matrikon OPC.
melsec-q-tcp	ICS	Get the CPU information from a Mitsubishi Electric Q Series PLC.
melsec-q-udp	ICS	Get the CPU information from a Mitsubishi Electric Q Series PLC.
modbus	ICS	Grab the Modbus device information via functions 17 and 43.
omron-tcp	ICS	Gets information about the Omron PLC.
pcworx	ICS	Gets information about the Omron PLC.
plc5	ICS	Rockwell Automation. Customers are encouraged to migrate from the PLC-5 Control System to the ControlLogix Control System.
proconos	ICS	Gets information about the PLC via the ProConOs protocol.
realport	ICS	Get the banner for the Digi Realport device
redlion-crimson3	ICS	A fingerprint for the Red Lion HMI devices running CrimsonV3
s7	ICS	Communicate using the S7 protocol and grab the device identifications.
secure-fox	ICS	Grabs a banner for proprietary FOX protocol by Tridium
toshiba-pos	ICS	Grabs device information for the IBM/ Toshiba 4690.
vertx-edge	ICS	Checks whether the device is running the VertX/ Edge door controller.
wdbrpc	ICS	Checks whether the WDB agent (used for debugging) is enabled on a VxWorks device.

Table 4: Shodan modules communicating with ICS devices

5.3 Data Sampling

As Shodan executes and captures approximately a quarter billion scans every week it was important that we get a smaller sample of the data within Shodan. For the purpose of this research we are mainly concerned with ICS devices protocols. The first four weeks of 2017 gave us 1,187,384 scans in the US. These ICS scans included 385,152 unique IP addresses and 498,022 IP / Port combinations. In order to understand the information further we looked at the ICS scans per state. Table 5: Top 20 ICS scans per state shows the top 20 states by the number of ICS Shodan data objects that were captured. Since we are located in Arizona most of the following information pertains to information gathered to coincide with the Arizona scans.

State	Total Scans	ICS Module Count
CA	141456	27
DE	83568	27
TX	81281	27
NY	16857	26
CO	14195	26
NJ	12762	26
IL	11107	27
VA	10068	27
UT	10053	26
FL	9640	27
WY	9591	25
MI	8072	27
GA	6555	26
MO	6455	26
AZ	6253	26
MA	6153	26
WA	5980	27
OR	4962	27
NC	4586	27
PA	3671	27

Table 5: Top 20 ICS scans per state

5.4 Feature Selection & Analysis

Some interesting information I found from the US ICS scans are:

- 114 scans tagged as honeypots (65 unique IPs)
- 2869 scans with associated vulnerabilities (1189 unique IPs)

Future work can include adding the tags Shodan put on the scan objects into the visualizations to try to confirm our suspicions that IPs with many (> 100) ports open are honeypots or NAT servers.

We would also like to highlight the IPs with vulnerabilities. For example, several were found to still be susceptible to the Heartbleed bug.

5.4.1 Reverse DNS & IP Locations

Other pieces of interesting information that can be used to supplement the Shodan data is trying to match the IP location to the country of the domain found in the reverse DNS lookup. Table 6: Scans where country code of IP and domain differ shows some of the data that can be queried from our database.

scanID	ip	port	shodanModule	IP Country	Domain Country
58633c6c27105869fa71	1482898540	10000	https-simple	GB	DE
b215746627105869fa71	2987750502	10000	https-simple-new	NL	IT
52c6556f01d15869fa71	1388729711	465	smtps	DE	AT
481d46ad1f905869fa71	1209878189	8080	http	US	BR
a2dee31d08235869fa72	2732516125	2083	https	US	IN
b91f9d2f024b5869fa72	3105856815	587	smtp	PT	CO
94fbaac400195869fa72	2499521220	25	smtp	DE	AT
2e692b2000165869fa72	778644256	22	ssh	FR	EU
5413d87c1f905869fa72	1410586748	8080	http	DE	AG
c14664dd08275869fa72	3242616029	2087	https-simple-new	IT	EU
d520312000b35869fa72	3575656736	179	bgp	DK	EU
534dc02d1d7b5869fa72	1397604397	7547	http-simple-new	EU	CH
bca5b00e00195869fa73	3164975118	25	smtp	FR	BR
a484ebb800505869fa74	2760174520	80	http	FR	EU
d83b151700165869fa74	3627750679	22	ssh	US	BR
b9133496008f5869fa74	3105043606	143	imap	DE	EU
5159c43700195869fa74	1364837431	25	smtp	DE	EU
422dede300355869fa74	1110306275	53	dns-udp	US	BR
5cdea2b800165869fa75	1558094520	22	ssh	FR	EU
b98861de00b35869fa75	3112722910	179	bgp	BG	MX

Table 6: Scans where country code of IP and domain differ

Future research can be done to determine if there is a correlation between mismatched country codes and malicious actors, or if it can be used as a feature to identify organizations.

5.4.2 Change in Scans over time

One of the biggest things we found is that there is no consistency in what is scanned within a time period. We had been working on the assumption that the entire IPv4 address space was scanned for all ports and protocols every 2 weeks. We now know this is not exactly accurate. Table 7: Change in frequency of scans shows a comparison of scan counts.

shodanModule	August 2016		January 2017		Percent Change
	Count	Percent	Count	Percent	
iec-104	878600	99.48%	4566	0.52%	-99.48%
modbus	19280	71.02%	7866	28.98%	-59.20%
s7	877	66.95%	433	33.05%	-50.63%
knx	4	66.67%	2	33.33%	-50.00%
melsec-q-udp	380	64.30%	211	35.70%	-44.47%
realport	13222	62.37%	7977	37.63%	-39.67%
redlion-crimson3	8838	62.14%	5384	37.86%	-39.08%
codesys	18429	62.01%	11289	37.99%	-38.74%
fox	91131	60.88%	58559	39.12%	-35.74%
automated-tank-gauge	24792	58.34%	17707	41.66%	-28.58%
pcworx	8197	57.53%	6051	42.47%	-26.18%
melsec-q-tcp	9657	56.53%	7427	43.47%	-23.09%
omron-tcp	21437	54.55%	17860	45.45%	-16.69%
hart-ip-udp	36232	53.03%	32088	46.97%	-11.44%
wdbrpc	34197	52.31%	31177	47.69%	-8.83%
general-electric-srtp	24395	51.42%	23046	48.58%	-5.53%
ethernetip-udp	5567	50.65%	5424	49.35%	-2.57%
proconos	20664	50.20%	20496	49.80%	-0.81%
lantronix-udp	36053	49.82%	36311	50.18%	0.72%
ethernetip	29840	49.47%	30484	50.53%	2.16%
dnp3	214200	47.06%	240950	52.94%	12.49%
coap	4503	46.26%	5231	53.74%	16.17%
matrikon-opc	16916	44.82%	20823	55.18%	23.10%
secure-fox	2935	44.06%	3726	55.94%	26.95%
bacnet	16413	37.75%	27070	62.25%	64.93%
vertx-edge	13799	33.27%	27673	66.73%	100.54%
kamstrup	4	22.22%	14	77.78%	250.00%
plc5	147162	22.14%	517619	77.86%	251.73%

Table 7: Change in frequency of scans

Looking at the scan data between August of 2016 (2016 ISO weeks 31-34) and January of 2017 (2017 ISO weeks 01-04) we can see that there are some drastic changes in the number of scans performed on iec-104, Modbus, plc5 and other ICS protocols. Again, another anomaly to be further investigated.

6 Data Visualization

6.1 Changes Over Time

We can see visual evidence that the scans performed on ICS devices in Venezuela between August 2016 and January 2017 have a large discrepancy [Afarin, 2017]. Figure 3: Venezuela ICS devices Aug 2016 [Afarin, 2017] below shows that in August 2016 there were 1415 IP addresses that had positive results when being scanned by the iec-104 Shodan module. Figure 4: Venezuela ICS devices Jan 2017 [Afarin, 2017] shows that by January 2017 this number had decreased to 6 IP addresses.

August 2016 (ISO Week 31 – 34)

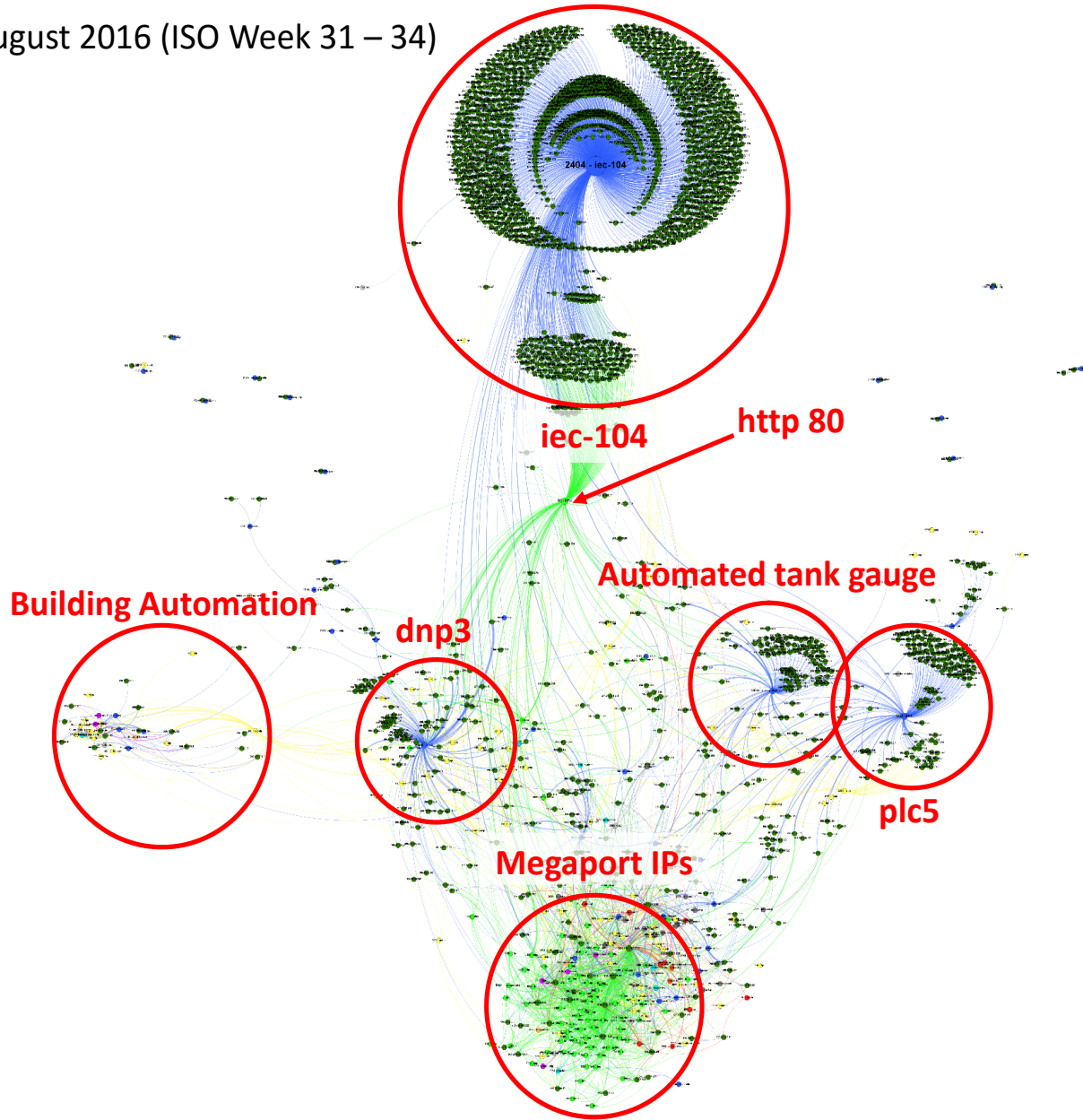


Figure 3: Venezuela ICS devices Aug 2016 [Afarin, 2017]

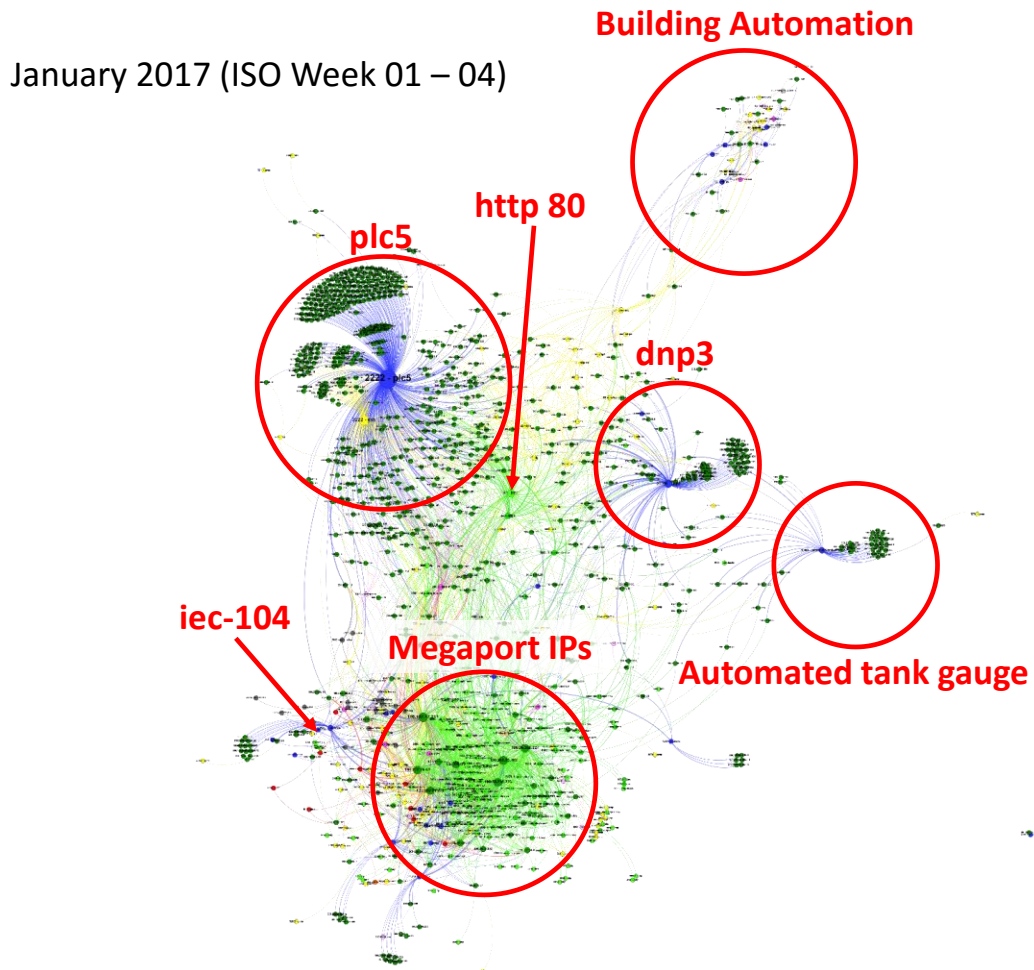


Figure 4: Venezuela ICS devices Jan 2017 [Afarin, 2017]

Using both the visual and the tabular data show the inconsistency of the Shodan scans over time. It is more readily apparent in the visual representation with the tabular data working well to give the details of what is happening.

6.2 Finding Malicious Protocols

For a semester project we looked at plc5 devices in Arizona as plc5 is known to be insecure. The visualizations show that all the IP addresses that had malware and Trojans on them belong to the same cluster or neighborhood. Using modularity in Gephi we are able to see that all the malicious protocols are in the orange colored neighborhood. Figure 5: Arizona PLC5 IPs by Modularity shows the neighborhoods created using modularity.

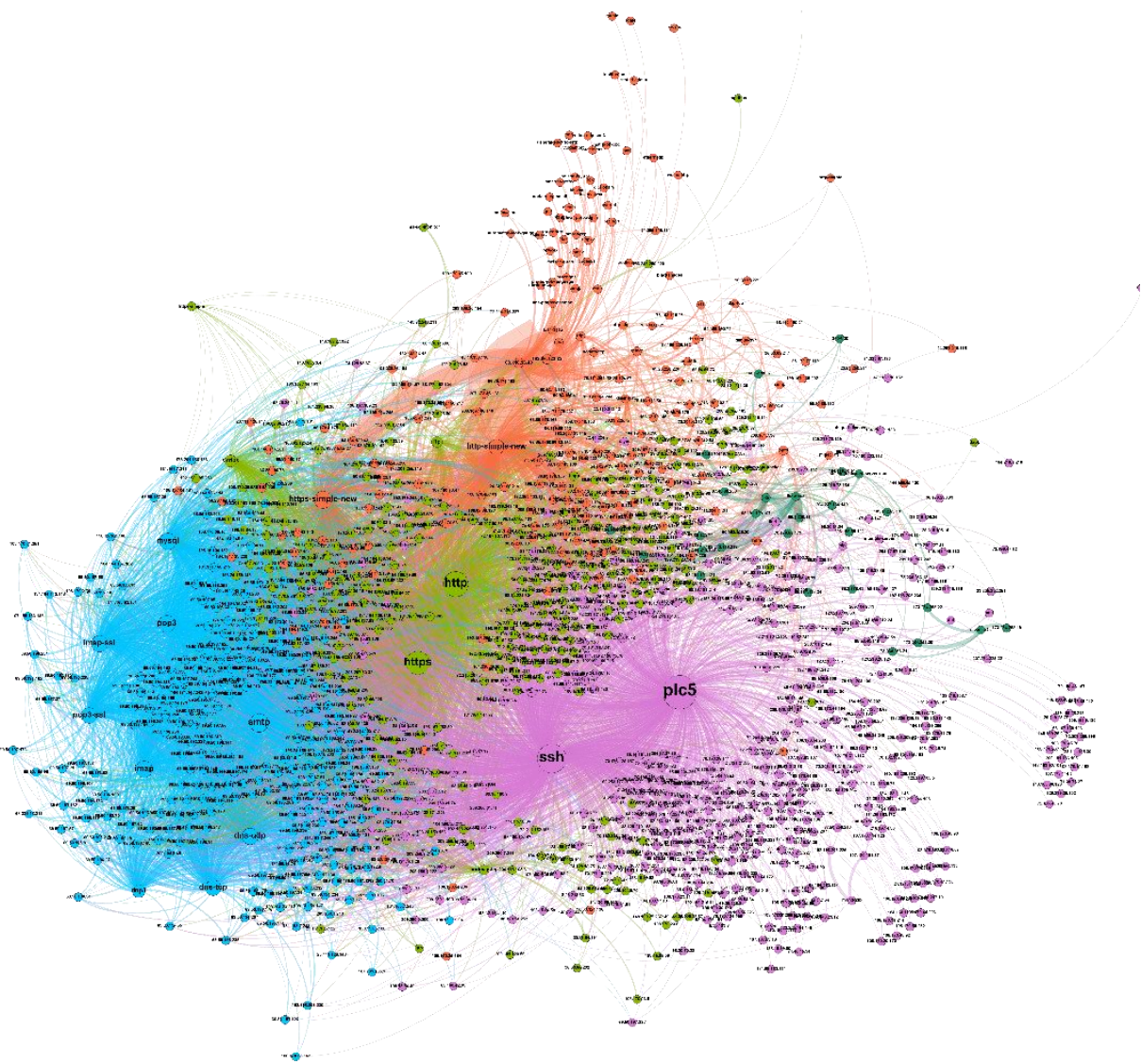


Figure 5: Arizona PLC5 IPs by Modularity

In Figure 6: Arizona plc5 IPs by protocol type the blue nodes are ICS protocols and the dark green nodes are IP addresses. The IP address nodes point to all the protocols that are running on them. The dark red nodes are malicious protocols: All located in the upper part of the image.

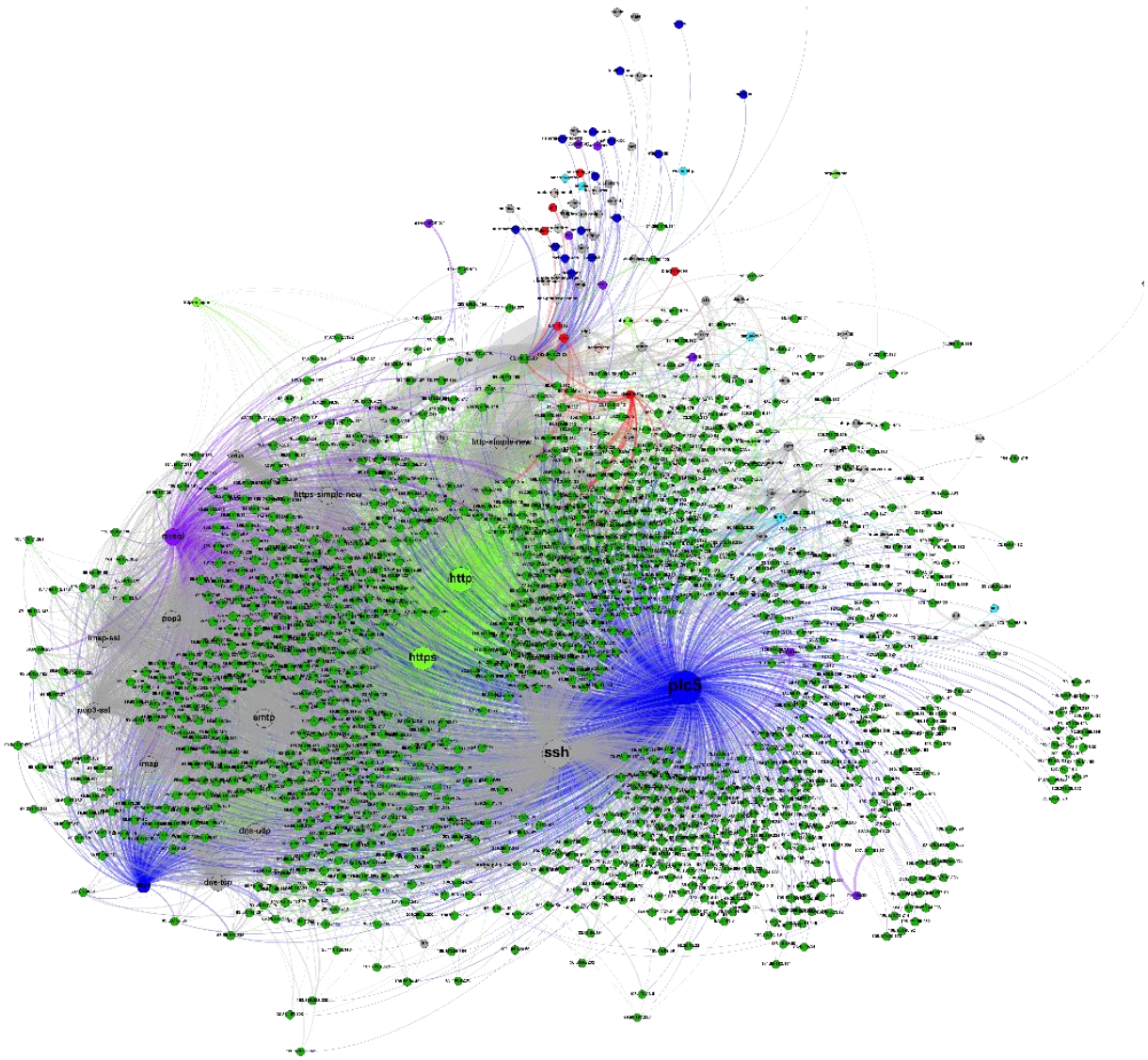


Figure 6: Arizona plc5 IPs by protocol type

Future work will be used to see if we can create a classification algorithm to determine if we can find malicious IPs on a network

6 Conclusion

As we can see, Shodan data has many uses but there are significant limitations to what it can tell us as well. There is no guarantee that the entirety of the internet is scanned within any specific time frame although on well-known ports like port 80 it appears that the majority of the internet would be captured within 3 to 4 weeks [Bodenheim, 2014]. Rohrmann's comparisons to his own nMap scans have shown that there are IPs that have ports open that are not showing up on Shodan within a timeframe [Rohrmann, 2017]. For example, we have seen 9000 scans in a week for an IP range on a certain port to just find out that it was a single unique IP that was being scanned continuously.

Looking at Shodan data from time periods does not show change in the internet structure. It only shows what was scanned. It also does not show that something that had been there in the past is no longer there. Further research needs to be done to find how long it takes to truly scan the whole internet as well as how often Shodan truly repeats scans. It is important to know if I should look at two weeks, a month or three months to get an accurate understanding of what is on the internet. A tool also needs to be created to verify if Shodan data is still accurate over time.;

It does do a large amount of correlating NSLookups, Whois, and IP Geolocation data with the scans that are performed. Further work needs to be done in many areas as discussed previously in this paper.

APPENDIX A – Shodan Data

Shodan data is stored in JSON format. There is one scan to a line and the JSON object can vary greatly depending on when and how the scan was completed. This section will cover the known parts of a Shodan scan and try to explain where unknown fields are located. The Shodan website provides a sparse banner specification for their REST and Streaming APIs at <https://developer.shodan.io/api/banner-specification>. The information on the page does not show a complete specification for what is in the raw scan files. Below I will try to explain the known keys that we have found of interest.

Main Properties

Property Name	Data Type	Description
ip	[Integer]	The IP address of the host as an integer.
ip_str	[String]	The IP address of the host as a string.
IPv6	[String]	The IPv6 address of the host as a string. If this is present then the "ip" and "ip_str" fields won't be.
asn	[String]	The autonomous system number (ex. "AS4837").
port	[Integer]	The port number that the service is operating on.
timestamp	[String]	The timestamp for when the banner was fetched from the device in the UTC time zone. Example: "2014-01-15T05:49:56.283713"
transport	[String]	Either "udp" or "tcp" to indicate which IP transport protocol was used to fetch the information
os	[String]	The operating system that powers the device.
location	[Object]	An object containing all of the location information for the device.

area_code	[Integer]	The area code for the device's location. Only available for the US.
city	[String]	The name of the city where the device is located.
country_code	[String]	The 2-letter country code for the device location.
country_code3	[String]	The 3-letter country code for the device location.
country_name	[String]	The name of the country where the device is located.
dma_code	[Integer]	The designated market area code for the area where the device is located. Only available for the US.
latitude	[Double]	The latitude for the geolocation of the device.
longitude	[Double]	The longitude for the geolocation of the device.
postal_code	[String]	The postal code for the device's location.
region_code	[String]	The name of the region where the device is located.
data	[String]	Contains the banner information for the service.
org	[String]	The name of the organization that is assigned the IP space for this device.
isp	[String]	The ISP that is providing the organization with the IP space for this device. Consider this the "parent" of the organization in terms of IP ownership.
hostnames	[String[]]	An array of strings containing all of the hostnames that have been assigned to the IP address for this device.
domains	[String[]]	An array of strings containing the top-level domains for the hostnames of the device. This is a utility property in case you want to filter by TLD instead of subdomain. It is smart enough to handle global TLDs with several dots in the domain (ex. "co.uk").

Table 8: Main Shodan Object Properties

Properties not in this table are optional and may not be found within the JSON object. This is the data available as of the date this was written.

Opt Properties

Property Name	Data Type	Description
opts	[Object]	Contains experimental and supplemental data for the service. This can include the SSL certificate, robots.txt and other raw information that hasn't yet been formalized into the Banner Specification.
uptime	[Integer]	The number of minutes that the device has been online.
link	[String]	The network link type. Possible values are: "Ethernet or modem", "generic tunnel or VPN", "DSL", "IPIP or SIT", "SLIP", IPsec or GRE, "VLAN", "jumbo Ethernet", "Google", "GIF", "PPTP", "loopback", "AX.25 radio modem".
title	[String]	The title of the website as extracted from the HTML source.
html	[String]	The raw HTML source for the website.
product	[String]	The name of the product that generated the banner.
version	[String]	The version of the product that generated the banner.
devicetype	[String]	The type of device (webcam, router, etc.).
info	[String]	Miscellaneous information that was extracted about the product.
cpe	[String]	The relevant Common Platform Enumeration for the product or known vulnerabilities if available. For more information on CPE and the official dictionary of values visit the CPE Dictionary.

Table 9: Optional Shodan Object Properties

SSL Properties

Property Name	Data Type	Description
ssl	[Object]	If the service uses SSL, such as HTTPS, then the banner will also contain a property called "ssl":
cert	[Object]	The parsed certificate properties that includes information such as when it was issued, the SSL extensions, the issuer, subject etc.
cipher	[Object]	Preferred cipher for the SSL connection
chain	[Array]	An array of certificates, where each string is a PEM-encoded SSL certificate. This includes the user SSL certificate up to its root certificate.
dhparams	[Object]	The Diffie-Hellman parameters if available: "prime", "public_key", "bits", "generator" and an optional "fingerprint" if we know which program generated these parameters.
versions	[Array]	A list of SSL versions that are supported by the server. If a version isn't supported the value is prefixed with a "-". Example: ["TLSv1", "-SSLv2"] means that the server supports TLSv1 but doesn't support SSLv2.

Table 10: SSL Object Properties in Shodan Object

All other properties found within the JSON object are not been officially defined. When these properties are explicitly parsed from the object for storage they will be defined further.

APPENDIX B – Shodan Tables

The Shodan database is designed to be able to store all information from the Shodan data object and be future proof to insure that all data is stored. This does not mean that all data is usable in the current form for research purposes. Because of this the database is broken into distinct parts that can be parsed and loaded depending on the information needed for research. There is a color coded ERD of the database in [Appendix B: Shodan Database ERD](#).

A short breakdown of these sections follows:

Core Data	This includes the primary scan data of ip, port, timestamp, Shodan module, location, etc.
Banner Data	Storage of banner data from the Shodan object
Common Properties	Common properties needed for research, i.e., os, tags, product, and version
Host / Domain Data	Host and domain information from doing an nslookup on scan data
HTML Data	Storage of HTML data from Shodan object
IP Address History	Timestamped storage of org and isp for an ip
Opt Data	Data from opt object (does not include object if vulns field is present)
Vulnerability Opt Data	Data from opt object if vulns field is present
SSL Data	Storage of the SSL object
Properties	All other properties not covered above

Table 11: Parser and Data store Sections

These ten sections of the Shodan object are utilized throughout the parsing and loading of Shodan data into the database. They will be described more deeply in the following sections of this document. For now we will go into the database in more detail.

Information Tables

This section provides information about informational tables that are not filled by the parsing of the JSON files provided by Shodan. These tables are used to supplement the information that we currently have in Shodan to investigate the information further.

ShodanModules

This table describes the current understanding of the Shodan module used to communicate with the port during a scan. The information in this table was acquired by researching the names of the modules online. This is not a definitive list or description of what each module does. It will need to be updated as more information and modules become available.

Field	Data Type	Description
shodanModule	VARCHAR PRIMARY KEY	Shodan Module as it comes from the JSON object.
shodanName	VARCHAR	Name from the Shodan API.
class	VARCHAR	Class given to the module.
shodanDescription	VARCHAR	The API description of the Shodan module
description	TEXT	Description from looking up the name online.

Table 12: Shodan Modules

The class field is to attempt to classify the devices or port purposes that are being communicated with. Currently there are 15 classes. They are as follow:

- ICS – Industrial control devices (may also contain SCADA)

- TORRENT – Bit torrent protocols including trackers
- Trojan – Communications with remote access Trojans (RATs) and others
- DB – Databases & DBMS. i.e., MySQL, MongoDB, Oracle, etc.
- IOT – internet of things devices. i.e., iKettle
- HTTP – uses HTTP
- HTTPS – uses HTTPS
- AUTH – authentication protocols
- PRINT – printers and print spools
- GAME – game servers. i.e., Minecraft
- VOIP – voice over IP
- TEST – test communication, this includes sending nothing or just a new line character
- REMOTE – remote access protocols. i.e., RDP
- Malware – Communicating with known malware
- TOR – Tor communications

There are many standard internet protocols (imap, dns, dhcp, etc.) that are not classed at this point in time. Some thought will need to go into a classification system.

Core Data

Core data is the basic data for when a scan took place. The location and whois information is included in the Locations and IPAddresses tables. The main scan table is named ipv4_scans_WW where WW is the ISO week that the scan took place. IPv6 information is ignored at this point in time. All tables with the _WW suffix will be of the ISO week. Queries over larger periods of time

can be done by using a union of queries for each week. Non weekly table were created for any information that was relatively constant and small.

IPv4_Scans_WW

The scanID from this table is used throughout the schema to indicate what scan the stored information is referring to. The scanID is create by concatenating the hex value of the ip, port, and timestamp.

Field	Data Type	Description
scanID	CHAR(20) PRIMARY KEY	Unique Identifier for a scan.
ip	INT	The IP address of the host as an integer.
port	INT	The port number that the service is operating on.
timestamp	INT	The UNIX timestamp of when scan took place
transport	SET	Either "udp" or "tcp" to indicate the IP transport protocol used
shodanModule	VARCHAR	Foreign Key to the ShodanModules table.
locationID	CHAR(32)	Foreign key to the Locations table.

Table 13: IPv4_Scans_WW

IPAddresses

This information is for the first time we saw the IP for that ISO year. Look to IPAddress_History_WW for all scan info.

Field	Data Type	Description
ip	INT PRIMARY KEY	The IP address of the host as an integer.
org	VARCHAR	The name of the organization that is assigned the IP space for this device.

isp	VARCHAR	The ISP that is providing the organization with the IP space for this device. Consider this the "parent" of the organization in terms of IP ownership.
------------	---------	--

Table 14: IPAddresses

Locations

Location give the relative location for the IP address of the scan. This table holds the location data.

The locationID is the primary key created using the concatenation of the formatted latitude and longitude.

Field	Data Type	Description
locationID	CHAR(32) PRIMARY KEY	Unique identifier for the location.
areaCode	INT	The area code for the device's location. Only available for the US.
city	VARCHAR	The name of the city where the device is located.
countryCode	CHAR(2)	The 2-letter country code for the device location.
countryCode3	CHAR(3)	The 3-letter country code for the device location.
countryName	VARCHAR	The name of the country where the device is located.
dmaCode	INT	The designated market area code for the area where the device is located. Only available for the US.
latitude	DECIMAL	The latitude for the geolocation of the device.
longitude	DECIMAL	The longitude for the geolocation of the device.
postalCode	VARCHAR	The postal code for the device's location.
regionCode	VARCHAR	The name of the region where the device is located.

Table 15: Locations

Banner Data

Banner data is data associated with the storing of banners retrieved during scans. Certain dates are removed from the banners such as modified and expires before being MDS'd and saved. This

allows us to have the same banner even if the current time and date are used for each scan. An MD5 of the date removed banner is used as a unique identifier of the banner.

Banners_WW

This table holds all the banners seen in this ISO week.

Field	Data Type	Description
bannerID	CHAR(32) PRIMARY KEY	Unique identifier for the banner.
banner	TEXT	Full text of the banner

Table 16: Banners_WW

Scan_Banner_WW

This table associated the banner with the scan that pulled it.

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
bannerID	CHAR(32)	Unique identifier for the banner.

Table 17: Scan_Banner_WW

Banner_Dates_WW

This table holds the dates that were removed from a scans banner.

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
dateType	ENUM	'Date', 'Expires', or 'Modified'
date	DATE	The date removed from the banner.

Table 18: Banner_Dates_WW

Common Properties

Common properties stores information for the following fields: os, version, product, tags.

Common properties can be expanded in the future as additional fields become relevant to the

research taking place. It is also recommended that we made need a crawler table to understand what the different crawlers responsibilities are.

Common_Properties_WW

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
property	VARCHAR	The name of the property in the JSON file.
data	TEXT	Data the property points to.

Table 19: Common_Properties_WW

Host / Domain Data

Holds the host and domain information retrieved with the scan.

Host_Dom_Properties_WW

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
property	VARCHAR	The name of the property in the JSON file. (‘hosts’, ‘domains’)
data	TEXT	Data the property points to.

Table 20: Host_Dom_Properties_WW

HTML Data

HTML data is data associated with the storing of html retrieved during scans. An MD5 of the html is used as a unique identifier.

HTML_WW

This table holds all the html seen in this ISO week.

Field	Data Type	Description
htmlID	CHAR(32) PRIMARY KEY	Unique identifier for the html.
html	TEXT	Full text of the html

Table 21: HTML_WW

Scan_HTML_WW

This table associated the html with the scan that pulled it.

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
htmlID	CHAR(32)	Unique identifier for the html.

Table 22: Scan_HTML_WW

IP Address History

Hold the information retrieved each time the IP address was seen. Used to check if it changes over time. May become deprecated if research shows that this never changes over time.

IPAddress_History_WW

Field	Data Type	Description
ip	INT	The IP address of the host as an integer.
timestamp	INT	The UNIX timestamp of when scan took place
org	VARCHAR	The name of the organization that is assigned the IP space for this device.
isp	VARCHAR	The ISP that is providing the organization with the IP space for this device. Consider this the "parent" of the organization in terms of IP ownership.

Table 23: IPAddress_History_WW

Opt Data

All the properties in the opt object within the JSON object. If there is a property name 'vulns' the information is not stored.

Opt_Properties_WW

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
property	VARCHAR	The name of the property in the JSON file.
data	TEXT	Data the property points to.

Table 24: Opt_Properties_WW

Vulnerability Opt Data

All the properties in the opt object within the JSON object if it contains a property name 'vulns'.

Opt_Prop_Vuln_WW

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
property	VARCHAR	The name of the property in the JSON file.
data	TEXT	Data the property points to.

Table 25: Opt_Prop_Vuln_WW

SSL Data

All the properties in the opt object within the JSON object. An MD5 of the stringified SSL object is used as a unique identifier.

Scan_SSL_WW

This table associated the SSL with the scan that pulled it.

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
sslID	CHAR(32)	Unique identifier for the ssl.

Table 26: Scan_SSL_WW

SSL_Properties_WW

Field	Data Type	Description
sslID	CHAR(32)	Unique Identifier for the ssl object.
property	VARCHAR	The name of the property in the JSON file.
data	TEXT	Data the property points to.

Table 27: SSL_Properties_WW

Properties

Properties stores all information found in the JSON object that is not stored in one of the other tables.

Properties_WW

Field	Data Type	Description
scanID	CHAR(20)	Unique Identifier for a scan.
property	VARCHAR	The name of the property in the JSON file.
data	TEXT	Data the property points to.

Table 28: Properties_WW

Appendix C – Shodan Database ERD

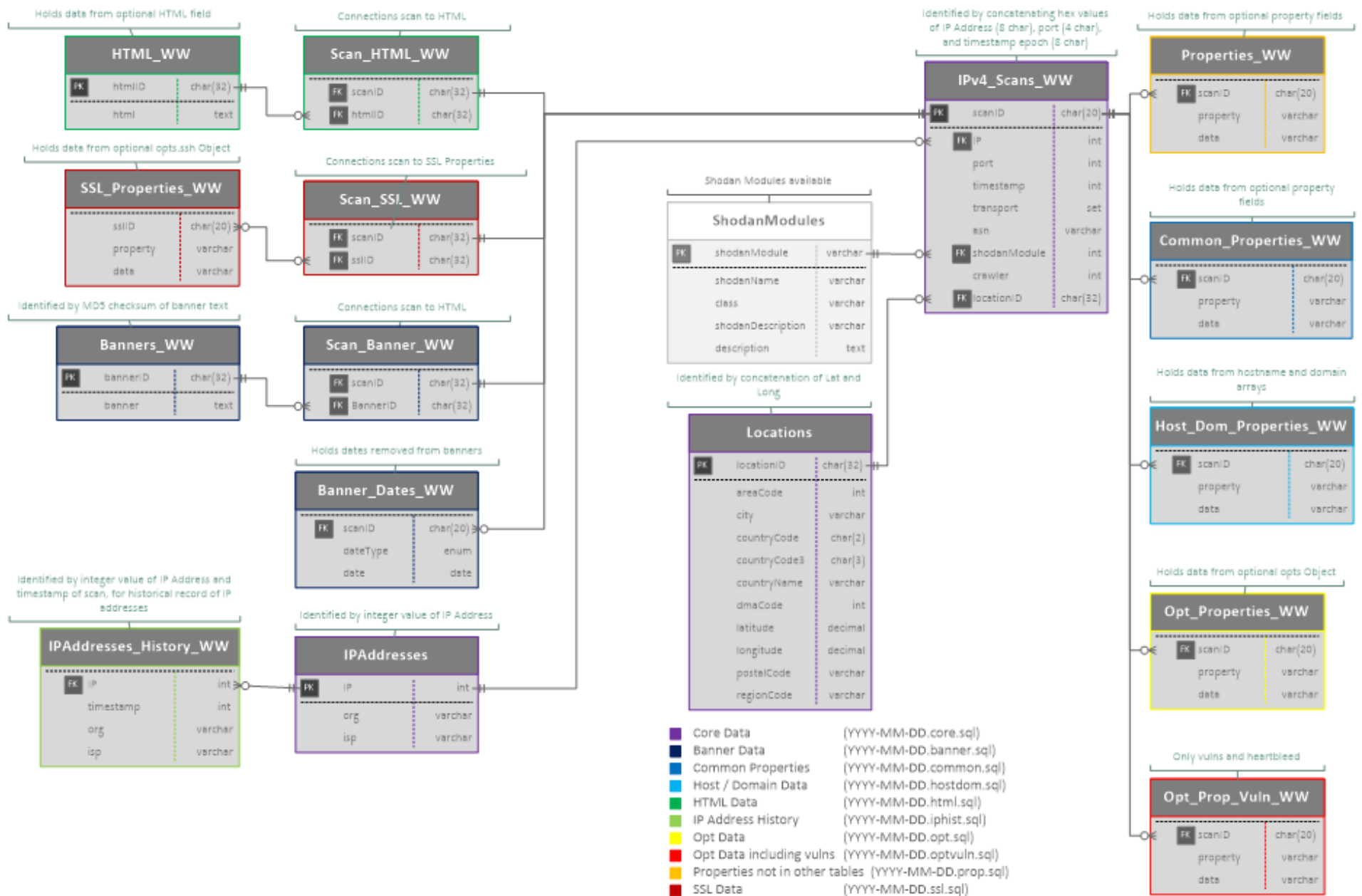


Figure 7: Shodan Database ERD

Appendix D – Shodan Modules and Classifications

Shodan Module	Class	Description
amqp	Unclassified	Grab information from an AMQP service
andromouse	Unclassified	Checks whether the device is running the remote mouse AndroMouse service.
apple-airport-admin	Unclassified	Check whether the device is an Apple AirPort administrative interface.
automated-tank-gauge	ICS	Get the tank inventory for a gasoline station.
bacnet	ICS	Gets various information from a BACnet device.
bgp	Unclassified	Checks whether the device is running BGP.
bitcoin	Unclassified	Grabs information about a Bitcoin daemon, including any devices connected to it.
bittorrent-tracker	TORRENT	Check whether there is a BitTorrent tracker running.
blackshades	Trojan	Determine whether a server is running a Blackshades C&C
cassandra	DB	Get cluster information for the Cassandra database software.
citrix-apps	Unclassified	This module attempts to query Citrix Metaframe ICA server to obtain a published list of applications.
clamav	Unclassified	Determine whether a server is running ClamAV
coap	ICS	Check whether the server supports the CoAP protocol
codesys	ICS	Grab a banner for Codesys daemons
couchdb	DB	HTTP banner grabbing module
dahua-dvr	IOT	Grab the serial number from a Dahua DVR device.
dhcp	Unclassified	Send a DHCP INFORM request to learn about the lease information from the DHCP server.
dht	Unclassified	Gets a list of peers from a DHT node.

Shodan Module	Class	Description
dicom	Unclassified	Checks whether the DICOM service is running.
dictionary	Unclassified	Connects to a dictionary server using the DICT protocol.
dnp3	ICS	A dump of data from a DNP3 outstation
dns-tcp	Unclassified	Try to determine the version of a DNS server by grabbing version.bind
dns-udp	Unclassified	Try to determine the version of a DNS server by grabbing version.bind
echo-udp	Unclassified	Checks whether the device is running echo.
epmd	Unclassified	Get a list of Erlang services and the ports they are listening on
ethernetip	ICS	Grab information from a device supporting EtherNet/IP over TCP
ethernetip-udp	ICS	Grab information from a device supporting EtherNet/IP over UDP
flux-led	IOT	Grab the current state from a Flux LED light bulb.
fox	ICS	Grabs a banner for proprietary FOX protocol by Tridium
frdm-1234	Unclassified	Checks for FRDM.
frdm-60007	Unclassified	Checks for FRDM.
ftp	Unclassified	Grab the FTP banner
gardasoft-vision	IOT	Grabs the version for the Gardasoft controller.
general-electric-srtp	ICS	Check whether the GE SRTP service is active on the device.
git	Unclassified	Check whether git is running.
gtp-v1	Unclassified	Checks whether the device is running a GPRS Tunnel.
hart-ip-udp	ICS	Checks whether the IP is a HART-IP gateway.
hbase	DB	Grab the status page for HBase database software.
hbase-old	DB	Grab the status page for old, deprecated HBase database software.

Shodan Module	Class	Description
hifly	IOT	Checks whether the HiFly lighting control is running.
http	HTTP	HTTP banner grabbing module
http-check	HTTP	HTTP banner grabbing module for Supermicro servers
http-simple	HTTP	Grabs the HTTP banner for a server but doesnt grab robots or anything else.
http-simple-new	HTTP	Grabs the HTTP banner for a server but doesnt grab robots or anything else.
http-supermicro	HTTP	HTTP banner grabbing module for Supermicro servers
https	HTTPS	HTTPS banner grabbing module
https-simple	HTTPS	HTTPS banner grabber only (no robots, sitemap etc.)
https-simple-new	HTTPS	HTTPS banner grabber only (no robots, sitemap etc.)
ibm-db2-das	DB	Grab basic information about the IBM DB2 Database Server.
ibm-nje	Unclassified	Check whether the z/OS Network Job Entry service is running.
idevice	IOT	Connects to an iDevice and grabs the property list.
iec-104	ICS	Banner grabber for the IEC-104 protocol.
iec-61850	Unclassified	MMS protocol
ike	Unclassified	Checks whether a device is running a VPN using IKE.
ike-nat-t	Unclassified	Checks whether a device is running a VPN using IKE and NAT traversal.
ikettle	IOT	Check whether the device is a coffee machine/ kettle.
imap	Unclassified	Get the welcome message of the IMAP server
imap-ssl	Unclassified	Get the welcome message of the secure IMAP server
ipmi	Unclassified	Checks whether a device is running IPMI remote management software.
java-rmi	Unclassified	Check whether the device is running Java RMI.

Shodan Module	Class	Description
kamstrup	ICS	Kamstrup Smart Meters
kerberos	AUTH	Checks whether a device is running the Kerberos authentication daemon.
kilerrat	Trojan	Determine whether a server is running a KilerRAT C&C
knx	ICS	Grabs the description from a KNX service.
lantronix-udp	ICS	Attempts to grab the setup object from a Lantronix device.
ldap	AUTH	LDAP banner grabbing module
ldap-udp	AUTH	CLDAP banner grabbing module
ldaps	AUTH	LDAPS banner grabbing module
lifx	TORRENT	Check whether there is a BitTorrent tracker running.
line-printer-daemon	PRINT	Get a list of jobs in the print queue to verify the device is a printer.
matrikon-opc	ICS	Checks whether the device is running Matrikon OPC.
mdns	Unclassified	Perform a DNS-based service discovery over multicast DNS
melsec-q-tcp	ICS	Get the CPU information from a Mitsubishi Electric Q Series PLC.
melsec-q-udp	ICS	Get the CPU information from a Mitsubishi Electric Q Series PLC.
memcache	Unclassified	Get general information about the Memcache daemon
minecraft	GAME	Gets the server status information from a Minecraft server
modbus	ICS	Grab the Modbus device information via functions 17 and 43.
mongodb	DB	Collects system information from the MongoDB daemon.
moxa-nport	IOT	Attempts to grab information from Moxna Nport devices.
mqtt	IOT	Grab a list of recent messages from an MQTT broker.
ms-sql-monitor	DB	Pings an MS-SQL Monitor server

Shodan Module	Class	Description
mumble-server	VOIP	Grabs the version information for the Mumble service (Mumble server)
munin	Unclassified	Check whether a Munin node is active and list its plugins
mysql	DB	Grabs the version of the running MySQL server
natpmp	Unclassified	Checks whether NAT-PMP is exposed on the device.
netbios	Unclassified	Grab NetBIOS information including the MAC address.
netmobility	Unclassified	Checks whether the device is a NetMobility.
newline-tcp	TEST	Connect to a server with TCP and send a newline.
newline-udp	TEST	Connect to a server with UDP and send a newline.
njrat	Trojan	Determine whether a server is running a njRAT C&C
nntp	Unclassified	Get the welcome message of a Network News server
nodata-tcp	TEST	Connect to a server without sending any data and store whatever it returns.
nodata-tcp-small	TEST	NULL
nodata-tcp-ssl	TEST	Connect to a server using SSL and without sending any data.
ntp	Unclassified	Get a list of IPs that NTP server recently saw and try to get version info.
nuclear-rat	Trojan	Checks whether the device is a C2 for Nuclear RAT.
omron-tcp	ICS	Gets information about the Omron PLC.
opc-ua	Unclassified	Grab a list of nodes from an OPC UA service
open-tcp	TEST	Checks whether a port is open and nothing else.
oracle-tns	DB	Check whether the Oracle TNS Listener is running.
pcanywhere-status	REMOTE	Asks the PC Anywhere status daemon for basic information.
pcworx	ICS	Gets information about the Omron PLC.

Shodan Module	Class	Description
plc5	ICS	Rockwell Automation. Customers are encouraged to migrate from the PLC-5 Control System to the ControlLogix Control System.
poison-ivy-rat	Trojan	Checks whether the device is running Poison Ivy.
pop3	Unclassified	Grab the POP3 welcome message
pop3-ssl	Unclassified	Grab the secure POP3 welcome message
portmap-tcp	Unclassified	Get a list of processes that are running and their ports.
portmap-udp	Unclassified	Get a list of processes that are running and their ports.
postgresql	DB	Collects system information from the PostgreSQL daemon
pptp	Unclassified	Connect via PPTP
printer-job-language	PRINT	Get the current output from the status display on a printer
proconos	ICS	Gets information about the PLC via the ProConOs protocol.
qratt	Trojan	Determine whether a server is running a QRAT C&C
rdate	Unclassified	Get the time from a remote rdate server
rdp	REMOTE	RDP banner grabbing module
realport	ICS	Get the banner for the Digi Realport device
redis	DB	Redis banner grabbing module
redlion-crimson3	ICS	A fingerprint for the Red Lion HMI devices running CrimsonV3
riak	DB	Sends a ServerInfo request to Riak
rip	Unclassified	Checks whether the device is running the Routing Information Protocol.
rsync	Unclassified	Get a list of shares from the rsync daemon.
rtsp-tcp	Unclassified	Determine which options the RTSP server allows.

Shodan Module	Class	Description
s7	ICS	Communicate using the S7 protocol and grab the device identifications.
secure-fox	ICS	Grabs a banner for proprietary FOX protocol by Tridium
serialnumbered	Unclassified	Checks for other servers with the same serial number on the local network. AAAAAA is a dummy value.
sip	Unclassified	Gets the options that the SIP device supports.
smarter-coffee	IOT	Checks the device status of smart coffee machines.
smb	Unclassified	Grab a list of shares exposed through the Server Message Block service
smtp	Unclassified	Get basic SMTP server response
smtps	Unclassified	Grab a banner and certificate for SMTPS servers
snmp	Unclassified	Gets the sysDescr.0 MIB of the SNMP service.
ssh	Unclassified	Get the SSH banner, its host key and fingerprint
steam-a2s	GAME	Get a list of IPs that NTP server recently saw and try to get version info.
steam-dedicated-server-rcon	GAME	Checks whether an IP is running as a Steam dedicated game server with remote authentication enabled.
ta14-353a	Malware	Alert (TA14-353A). Targeted Destructive Malware. Original release date: December 19, 2014 Last revised: September 30, 2016.
tacacs	AUTH	Check whether the device supports TACACS+ AAA.
teamviewer	REMOTE	Determine whether a server is running TeamViewer
telnet	Unclassified	Telnet banner grabbing module
telnets	Unclassified	Telnet wrapped in SSL banner grabbing module
tenda-backdoor	Trojan	Firmware backdoor in some models of Tenda routers allow for remote command execution

Shodan Module	Class	Description
tor-control	TOR	Checks whether a device is running the Tor control service.
tor-versions	TOR	Checks whether the device is running the Tor OR protocol.
toshiba-pos	ICS	Grabs device information for the IBM/ Toshiba 4690.
ubiquiti-discover	Unclassified	Grabs information about the Ubiquiti-powered device
udpxy	Unclassified	Udpxy banner grabbing module
upnp	Unclassified	Collects device information via UPnP.
ventrilo	VOIP	Gets the detailed status information from a Ventrilo server.
vertx-edge	ICS	Checks whether the device is running the VertX/ Edge door controller.
voldemort	DB	Pings the Voldemort database.
wdbrpc	ICS	Checks whether the WDB agent (used for debugging) is enabled on a VxWorks device.
wemo-http	IOT	Connect to a Wemo Link and grab the setup.xml file
x11	Unclassified	Connect to X11 w/ no auth and grab the resulting banner.
xmpp	Unclassified	Sends a hello request to the XMPP daemon
xtremerat	Trojan	XtremeRAT: Remote Access Trojan or Remote Administration Tool.
yahoo-smarttv	IOT	Checks whether the device is running the Yahoo Smart TV device communication service.
zookeeper	Unclassified	Grab statistical information from a Zookeeper node

Table 29: Shodan Modules

Appendix D – Shodan Modules and Classifications

parseShodanFile.py

```
'''
File name: parseShodanFile.py
Author: Vincent J Ercolani
Date created: 11/10/2015
Date last modified: 06/13/2016
Python Version: 3.5.1 64bit
Requires: shodan_sql_import.py
         from shodan_sql_import import processSQLImportFile

This program parses gzipped shodan files into csv files for import into MySQL.

usage: parseShodanFile.py [-h] -g GZIPFILE [-s SKIP] [-u UPDATE] [-f FLUSHFREQ] [-v]

required arguments:
  -g GZIPFILE, --gzipfile GZIPFILE  Full path of the gzipfile to be parsed

optional arguments:
  -h, --help                        show this help message and exit
  -s SKIP, --skip SKIP              Number of lines to skip before parsing
  -u UPDATE, --update UPDATE       Print update every [x] number of lines in file
  -f FLUSHFREQ, --flushfreq FLUSHFREQ Number of times through UPDATE lines to flush buffers
  -v, --verbose                    Display verbose output of parsing actions
'''
import argparse
import json
import gzip
import re
import os
from timeit import default_timer as timer
from shodan_sql_import import processSQLImportFile
from hashlib import md5
from datetime import datetime
import time
import csv
```

```

import io

# initialize timer
timer()

# Get script arguments
parser = argparse.ArgumentParser()
parser.add_argument('-g', '--gzipfile', help='Full path of the gzipfile to be parsed',
                    type=str, required=True)
parser.add_argument('-s', '--skip', help='Number of lines to skip before parsing',
                    type=int, default=0)
parser.add_argument('-u', '--update', help='Print update every [x] number of lines in file',
                    type=int, default=10000)
parser.add_argument('-f', '--flushfreq', help='Number of times through update lines to flush buffers',
                    type=int, default=8)
parser.add_argument('-v', '--verbose', help='Display verbose output of parsing actions',
                    action='store_true', default=False)
args = parser.parse_args()

#initialize global variables
count = 0
flushcount = 0
maxline = 3000000000

(path,basename) = os.path.split(args.gzipfile)
# Create file and path that we will move gzip file to upon completion
comppath = os.path.abspath(os.path.join(path, '..\\JSON_Complete'))
if not os.path.exists(comppath):
    try:
        os.makedirs(comppath)
    except OSError as exc: # Guard against race condition
        if exc.errno != errno.EEXIST:
            raise
mvfile = os.path.join(comppath,basename)

# Create output directory for the csv files
basename = basename.split('.')[0]
path = os.path.join(path,basename+'_parsed')
if not os.path.exists(path):
    try:
        os.makedirs(path)

```

```

except OSError as exc: # Guard against race condition
    if exc.errno != errno.EEXIST:
        raise

rejectout = open(os.path.join(path, basename + '__REJECT__.json'), 'w+', encoding='utf-8')
errorout = open(os.path.join(path, basename + '__ERROR__.log'), 'w+', encoding='utf-8')
failout = open(os.path.join(path, basename + '__FAIL__.json'), 'w+', encoding='utf-8')
log = open(os.path.join(path, basename + '.log'), 'w+', encoding='utf-8')

# hashmap of counters
counter = {}

# hashmaps to insure unique items
banners = {}
htmls = {}
locations = {}
ips = {}
ssls = {}

# hashmap for outfiles
out = {}
files = {}

# Tables that json will be parsed into
tables = ['scan', 'banner', 'banner_date', 'ip', 'location', 'html', 'scan_html',
          'properties', 'opts', 'ssl', 'scan_ssl', 'host_dom']

# Initialize headers to table files
header = {}
header['scan'] = ['scanID', 'IP', 'port', 'timestamp', 'transport', 'asn', 'shodanModule',
                 'crawler', 'bannerID', 'locationID']
header['banner'] = ['bannerID', 'banner']
header['banner_date'] = ['scanID', 'dateType', 'date']
header['ip'] = ['IP', 'org', 'isp']
header['location'] = ['locationID', 'areacode', 'city', 'countryCode', 'countryCode3', 'countryName',
                    'dmaCode', 'latitude', 'longitude', 'postaCode', 'regionCode']
header['html'] = ['htmlID', 'html']
header['scan_html'] = ['scanID', 'htmlID']
header['properties'] = ['scanID', 'property', 'data']
header['opts'] = ['scanID', 'property', 'data']
header['ssl'] = ['sslID', 'property', 'data']

```



```

header['scan_ssl']      = ['scanID','sslID']
header['host_dom']     = ['scanID','property','data']

def initFiles(basename):
    global out
    global files
    global counter

    for table in tables:
        quote = csv.QUOTE_MINIMAL
        if(table in ['banner','html','ssl','opts']):
            quote = csv.QUOTE_ALL

        filename = basename + '_' + table + '.csv'
        files[table] = open(os.path.join(path,filename), 'w', encoding='utf-8', newline='')
        out[table] = csv.writer(files[table], delimiter=',', quotechar='"',
                                escapechar='\\', quoting=quote, doublequote='false')

        out[table].writerow(header[table])

        counter[table] = 0

def xstr(s):
    if s is None:
        return ''
    return str(s)

def hexstr(num,len):
    return hex(num).split('x')[1].rjust(len,'0')

def createScanID(ip,port,timestamp):
    return hexstr(ip,8) + hexstr(port,4) + hexstr(int(time.mktime(timestamp.timetuple())) ,8)

def writeToStatsFile(basename, count):
    global counter

    processout = csv.writer(open('output_stats.csv', 'a', encoding='utf-8', newline=''), delimiter=',',
                            quotechar='"', escapechar='\\', quoting=csv.QUOTE_MINIMAL, doublequote='false')
    output = [basename, str(count)]
    for table in tables: output.append(str(counter[table]))
    output.append(str(timer()))

```

```

processout.writerow(output)

def processBanner(banner, scanID):
    global banners
    global counter

    bannerDates = re.findall('\w+?: \w/\w/\w, \d/\d \w/\w/\w \d/\d/\d/\d \d/\d:\d/\d:\d/\d \w/\w/\w', banner)
    banner = re.sub('\w+?: \w/\w/\w, \d/\d \w/\w/\w \d/\d/\d/\d \d/\d:\d/\d:\d/\d \w/\w/\w', '', banner)
    for dates in bannerDates:
        (dateType,date) = dates.split(': ')
        try:
            out['banner_date'].writerow([scanID,dateType,
                                         '{0:%Y-%m-%d %X}'.format(datetime.strptime(date[5:-4],"%d %b %Y %X"))])
            counter['banner_date'] += 1
        except ValueError as e:
            out['banner_date'].writerow([scanID,dateType,'Unknown: ' + date])
            errorout.write("ValueError: {}\tLine: {}\tFile: {}\n".format(scanID,str(count),args.gzipfile))
            errorout.write("\t\t" + scanID + "\tDates: " + dates + "\tDate: " + date + '\n')
            errorout.write("\t\t" + str(e) + '\n')
            if args.verbose:
                print("ValueError: {}\tLine: {}\tFile: {}".format(scanID,str(count),args.gzipfile))
            pass

    digest = md5(banner.encode('utf-8')).hexdigest()

    if digest in banners: return digest
    banners[digest] = 1

    out['banner'].writerow([digest,banner.encode('unicode_escape').decode('utf-8')])

    counter['banner'] += 1
    return digest

def processLocation(scanObj):
    global locations
    global counter

    location = scanObj.get('location')
    if(location is None): return None

    if(location.get('longitude') is None or location.get('latitude') is None):

```

```

        return None

locationID = "%016.11f" % location.get('latitude') + "%016.11f" % location.get('longitude')

if locationID in locations: return locationID
locations[locationID] = 1

out['location'].writerow([locationID,xstr(location.get('city')),xstr(location.get('country_code')),
                        xstr(location.get('country_code3')),xstr(location.get('country_name')),
                        xstr(location.get('area_code')),xstr(location.get('dma_code')),
                        xstr(location.get('latitude')),xstr(location.get('longitude')),
                        xstr(location.get('postal_code')),xstr(location.get('region_code'))])
counter['location'] += 1
return locationID

def processShodan(shodan):
# breakdown shodan obj into module and crawler
module = None
crawler = None
if (shodan is not None):
    module = shodan.get('module')
    crawler = shodan.get('crawler')

    return (module,crawler)

def processIP(ip,scanObj):
global ips
global counter

if ip in ips: return
ips[ip] = 1
out['ip'].writerow([xstr(ip),xstr(scanObj.get('org')),xstr(scanObj.get('isp'))])
counter['ip'] += 1

def processHtml(scanID,html):
global htmls
global counter

if(html is None): return

try:

```

```

digest = md5(html.encode('utf-8')).hexdigest()

if digest not in htmls:
    htmls[digest] = 1
    out['html'].writerow([digest,html.encode('unicode_escape').decode('utf-8')])
    counter['html'] += 1

    out['scan_html'].writerow([scanID,digest])
    counter['scan_html'] += 1
except MemoryError as e:
    errorout.write("HTML_MemoryError: {}\tLine: {}\tFile: {}\n".format(scanID,str(count),args.gzipFile))
    errorout.write("\t\t{}".format(str(e)))
    if args.verbose:
        print("HTML_MemoryError: {}\tLine: {}\tFile: {}".format(scanID,str(count),args.gzipfile))
        print("HTML_MemoryError: {}".format(str(e)))
    pass

def processSSL(scanID, sslObj):
    global ssls
    global counter

    if(sslObj is None): return

    digest = md5(json.dumps(sslObj).encode('utf-8')).hexdigest()

    if digest not in ssls:
        ssls[digest] = 1
        processProp(digest, 'ssl', sslObj)

    out['scan_ssl'].writerow([scanID,digest])
    counter['scan_ssl'] += 1

def processProp(scanID, prop, data):
    global counter

    if (data is not None):
        if(type(data) is dict):
            processObj(scanID,prop,data)
        elif(type(data) is list):
            processList(scanID,prop,data)
        else:

```

```

        (table,prop) = prop.split('.',1)
        out[table].writerow([scanID,xstr(prop),xstr(data).encode('unicode_escape').decode('utf-8')])
        counter[table] += 1

def processList(scanID, table, obj):
    if (obj is not None and len(obj) > 0):
        for data in obj:
            processProp(scanID,table,data)

def processObj(scanID, table, obj):
    for key in obj:
        tmp = obj[key]
        processProp(scanID,table + '.' + key,tmp)

def processAdditionalProperties(scanID, scanObj):
    global counter

    for prop in scanObj:
        if(prop not in ['ip','ip_str','port','timestamp','transport','asn',
            'org', 'isp','_shodan','data','location']):
            if(prop == 'html'):
                processHtml(scanID,scanObj[prop])
            elif(prop == 'ssl'):
                processSSL(scanID,scanObj[prop])
            elif(prop == 'opts'):
                processProp(scanID,prop,scanObj[prop])
            elif(prop in ['hostnames','domains']):
                processProp(scanID,'host_dom.' + prop, scanObj[prop])
            else:
                processProp(scanID,'properties.' + prop,scanObj[prop])

def processScan(scanObj):
    global counter

    # set up initial scan variables
    ip = scanObj.get('ip')
    if (ip is None): return

    port = scanObj.get('port')
    if (port is None): return

```

```

timestamp = scanObj.get('timestamp')
if (timestamp is None): return

ptimestamp = None
try:
    ptimestamp = datetime.strptime(timestamp, '%Y-%m-%dT%H:%M:%S.%f')
except ValueError:
    try:
        ptimestamp = datetime.strptime(timestamp, '%Y-%m-%dT%H:%M:%S')
    except ValueError:
        raise

# create identifier for the scan
scanID = createScanID(ip,port,ptimestamp)

# banner will always be present so send banner
bannerID = processBanner(scanObj.get("data"), scanID)
# additional error checking needed for location send json object
locationID = processLocation(scanObj)

# breakdown shodan obj into module and crawler
(module,crawler) = processShodan(scanObj.get('_shodan'))

# process the ip address and parts
processIP(ip,scanObj)

out['scan'].writerow([xstr(scanID),xstr(ip),xstr(port),ptimestamp.strftime('%Y-%m-%d %H:%M:%S.%f'),
                        xstr(scanObj.get('transport')),xstr(scanObj.get('asn')),xstr(module),xstr(crawler),
                        bannerID, xstr(locationID)])
processAdditionalProperties(scanID, scanObj)

counter['scan'] += 1

def outputUpdate():
    global flushcount

    time = "%018.11f" % timer()
    stats = basename + "\t" + time + "\t" + str(count) + '\tbanners: ' + \
        str(counter['banner']) + '\tlocations: ' + str(counter['location']) + '\tscans: ' + \
        str(counter['scan'])
    print(stats)

```

```

log.write(stats + '\n')

flushcount += 1
if((flushcount%args.flushfreq) == 0):
    for table in tables: files[table].flush()
    rejectout.flush()
    errorout.flush()
    failout.flush()
    log.flush()

def processFile(file):
    global count

    initFiles(basename)

    for line in file:
        if count >= args.skip and len(line) < maxline and '"ipv6":' not in line:
            # parse the line into a json object
            try:
                processScan(json.loads(line))
            except ValueError as e:
                print("ValueError: ", e)
                errorout.write("ValueError: " + str(e) + '\n')
                failout.write(line)
                pass
            except MemoryError as e:
                print("processFile MemoryError: ", e)
                errorout.write("processFile MemoryError: " + str(e))
                failout.write(line)
                raise
            except (BaseException, KeyboardInterrupt) as e:
                print('Handling run-time error: ', e)
                errorout.write("processFile Exception: " + str(e))
                failout.write(line)
                raise
        elif len(line) >= maxline:
            rejectout.write(line)
        count += 1
    if (count%args.update) == 0:
        outputUpdate()

```

```
def main():
    try:
        with io.TextIOWrapper(gzip.GzipFile(args.zipfile,'r')) as file:
            processFile(file)
            os.rename(args.zipfile,mvfile)
    except IOError as e:
        print('Handling IOError: ', e)
    except BaseException as e:
        print('Handling run-time error: ', e)
        errorout.write("processFile Exception: " + str(e))
    finally:
        print('Last line completed: ' + str(count) + "\t" + str(timer()))
        log.write('Last line completed: ' + str(count) + "\t" + str(timer()))
        processSQLImportFile(basename, path)
        writeToStatsFile(basename, count)

main()
```


shodan_sql_import.py

```
'''
File name: shodan_sql_import.py
Author: Vincent J Ercolani
Date created: 06/01/2016
Date last modified: 06/13/2016
Python Version: 3.5.1 64bit

This program create import files for shodan Sinfo into MySQL.

usage: from shodan_sql_import import processSQLImportFile
'''
import datetime
import os

isoweek = None

def processSQLImportFile(basename, path):
    global isoweek
    (isoyear, isoweek, null) = datetime.datetime.strptime(basename, '%Y-%m-%d').isocalendar()
    isoweek = str(isoweek).rjust(2, '0')

    fullname = os.path.join(path, basename).replace('\\', '\\\\')

    _processHTMLImportFile(fullname, isoyear, basename)
    with open(fullname + '.sql', 'w') as sqlout:
        sqlout.write("CREATE DATABASE IF NOT EXISTS ShodanData_{};\n".format(isoyear))
        sqlout.write("USE ShodanData_{};\n".format(isoyear))
        sqlout.write("\n")
        _writeCreateDatabase(sqlout)
        _writeStatTable(sqlout)
        _writeStartLoad(sqlout)
        _writeLoadDataInfile(sqlout, 'IPAddresses', 'ip', 'IP,org,isp', fullname)
        _writeLoadDataInfile(sqlout, 'Locations', 'location',
            'locationID,areaCode,city,countryCode,countryCode3,countryName,' + \
            'dmaCode,latitude,longitude,postalCode,regionCode', fullname)
        _writeLoadDataInfile(sqlout, 'Banners', 'banner', 'bannerID,banner', fullname)
        _writeLoadDataInfile(sqlout, 'IPv4_Scans', 'scan',
            'scanID,IP,port,timestamp,transport,asn,shodanModule,crawler,' + \
```

```

        'bannerID,locationID',fullname)
    _writeLoadDataInfile(sqlout,'Banner_Dates','banner_date','scanID,dateType,date',fullname)
    _writeLoadDataInfile(sqlout,'Properties','properties','scanID,property,data',fullname)
    _writeLoadDataInfile(sqlout,'Opt_Properties','opts','scanID,property,data',fullname)
    _writeLoadDataInfile(sqlout,'SSL_Properties','ssl','sslID,property,data',fullname)
    _writeLoadDataInfile(sqlout,'Scan_SSL','scan_ssl','scanID,sslID',fullname)
    _writeLoadDataInfile(sqlout,'Host_Dom_Properties','host_dom','scanID,property,data',fullname)
    _writeEndLoad(sqlout)
    _writeStats(sqlout,basename)

def _processHTMLImportFile(fullname,isoyear,basename):
    with open(fullname+'.html.sql','w') as sqlout:
        sqlout.write("USE ShodanData_{};\n\n".format(isoyear))
        _writeStartLoad(sqlout)
        _writeLoadDataInfile(sqlout,'HTML','html','htmlID,html',fullname)
        _writeLoadDataInfile(sqlout,'Scan_HTML','scan_html','scanID,htmlID',fullname)
        _writeEndLoad(sqlout)
        _writeUpdateStats(sqlout,basename)

def _writeLoadDataInfile(sqlout,table,prop,fields,fullname):
    orig = table
    if table not in ['IPAddresses','Locations']: table = table + '_' + isoweek
    sqlout.write("SET @CURTIME=CURRENT_TIMESTAMP();\n")
    sqlout.write("LOCK TABLES `{}` WRITE;\n".format(table))
    sqlout.write("ALTER TABLE `{}` DISABLE KEYS;\n".format(table))
    sqlout.write("LOAD DATA LOCAL INFILE '{}_{}.csv'\n".format(fullname,prop))
    sqlout.write(" IGNORE INTO TABLE {}\n".format(table))
    sqlout.write(" FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '\"'\n")
    sqlout.write(" ESCAPED BY '\\\\' LINES TERMINATED BY '\\r\\n'\n")
    sqlout.write(" IGNORE 1 LINES\n")
    sqlout.write(" ({}); \n".format(fields))
    sqlout.write("ALTER TABLE `{}` ENABLE KEYS;\n".format(table))
    sqlout.write("UNLOCK TABLES;\n")
    sqlout.write("SET @{}=UNIX_TIMESTAMP(CURRENT_TIMESTAMP())-UNIX_TIMESTAMP(@CURTIME);\n".format(orig))
    sqlout.write("SELECT      \"{}\"      as      mytable,      CURRENT_TIMESTAMP(),      @{}      as
elapsed_time;\n\n".format(table,orig))

def _writeCreateDatabase(sqlout):
    sqlout.write("CREATE TABLE IF NOT EXISTS IPAddresses (\n")
    sqlout.write(" IP INT UNSIGNED NOT NULL PRIMARY KEY,\n")
    sqlout.write(" org VARCHAR(255),\n")

```

```
sqlout.write(" isp VARCHAR(255)\n")
sqlout.write("ENGINE=MyISAM CHARACTER SET =utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Locations (\n")
sqlout.write(" locationID CHAR(32) NOT NULL PRIMARY KEY,\n")
sqlout.write(" areaCode INT UNSIGNED,\n")
sqlout.write(" city VARCHAR(255),\n")
sqlout.write(" countryCode CHAR(2),\n")
sqlout.write(" countryCode3 CHAR(3),\n")
sqlout.write(" countryName VARCHAR(255),\n")
sqlout.write(" dmaCode INT UNSIGNED,\n")
sqlout.write(" latitude DECIMAL(16,11),\n")
sqlout.write(" longitude DECIMAL(16,11),\n")
sqlout.write(" postalCode VARCHAR(20),\n")
sqlout.write(" regionCode VARCHAR(45)\n")
sqlout.write("ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Banners_{} (\n".format(isoweek))
sqlout.write(" bannerID CHAR(32) NOT NULL PRIMARY KEY,\n")
sqlout.write(" banner LONGTEXT\n")
sqlout.write("ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS IPv4_Scans_{} (\n".format(isoweek))
sqlout.write(" scanID CHAR(20) NOT NULL PRIMARY KEY,\n")
sqlout.write(" IP INT UNSIGNED NOT NULL REFERENCES IPAddresses,\n")
sqlout.write(" port INT UNSIGNED NOT NULL,\n")
sqlout.write(" timestamp TIMESTAMP NOT NULL,\n")
sqlout.write(" transport ENUM('tcp','udp'),\n")
sqlout.write(" asn VARCHAR(8),\n")
sqlout.write(" shodanModule VARCHAR(255),\n")
sqlout.write(" crawler VARCHAR(255),\n")
sqlout.write(" bannerID CHAR(32) NOT NULL REFERENCES Banners_{},\n".format(isoweek))
sqlout.write(" locationID CHAR(32) DEFAULT NULL REFERENCES Locations ON DELETE SET NULL\n")
sqlout.write("ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Banner_Dates_{} (\n".format(isoweek))
sqlout.write(" scanID CHAR(20) NOT NULL REFERENCES IPv4_Scans_{},\n".format(isoweek))
sqlout.write(" dateType VARCHAR(255) NOT NULL,\n")
sqlout.write(" date TIMESTAMP NOT NULL,\n")
sqlout.write(" PRIMARY KEY (scanID,dateType)\n")
sqlout.write("ENGINE=MyISAM CHARACTER SET=utf8;\n")
```

```

sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS HTML_{} (\n".format(isoweek))
sqlout.write("  htmlID  CHAR(32) NOT NULL PRIMARY KEY,\n")
sqlout.write("  html    LONGTEXT\n")
sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Scan_HTML_{} (\n".format(isoweek))
sqlout.write("  scanID  CHAR(20) NOT NULL REFERENCES IPv4_Scan_{},\n".format(isoweek))
sqlout.write("  htmlID  CHAR(32) NOT NULL REFERENCES HTML_{},\n".format(isoweek))
sqlout.write("  PRIMARY KEY (scanID,htmlID)\n")
sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Properties_{} (\n".format(isoweek))
sqlout.write("  scanID  CHAR(20) NOT NULL REFERENCES IPv4_Scan_{},\n".format(isoweek))
sqlout.write("  property VARCHAR(255) NOT NULL,\n")
sqlout.write("  data    TEXT,\n")
sqlout.write("  PRIMARY KEY (scanID,property)\n")
sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Opt_Properties_{} (\n".format(isoweek))
sqlout.write("  scanID  CHAR(20) NOT NULL REFERENCES IPv4_Scan_{},\n".format(isoweek))
sqlout.write("  property VARCHAR(255) NOT NULL,\n")
sqlout.write("  data    TEXT\n")
sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS SSL_Properties_{} (\n".format(isoweek))
sqlout.write("  sslID   CHAR(32) NOT NULL,\n")
sqlout.write("  property VARCHAR(255) NOT NULL,\n")
sqlout.write("  data    TEXT\n")
sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Scan_SSL_{} (\n".format(isoweek))
sqlout.write("  scanID  CHAR(20) NOT NULL REFERENCES IPv4_Scan_{},\n".format(isoweek))
sqlout.write("  sslID   CHAR(32) NOT NULL REFERENCES SSL_Properties_{},\n".format(isoweek))
sqlout.write("  PRIMARY KEY (scanID,sslID)\n")
sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")
sqlout.write("CREATE TABLE IF NOT EXISTS Host_Dom_Properties_{} (\n".format(isoweek))
sqlout.write("  scanID  CHAR(20) NOT NULL REFERENCES IPv4_Scan_{},\n".format(isoweek))
sqlout.write("  property VARCHAR(255) NOT NULL,\n")
sqlout.write("  data    TEXT\n")

```

```

sqlout.write("ENGINE=MyISAM CHARACTER SET=utf8;\n")
sqlout.write("\n")

def _writeStartLoad(sqlout):
    sqlout.write("SET @START_TIME=CURRENT_TIMESTAMP();\n")
    sqlout.write("SELECT @START_TIME;\n")
    sqlout.write("\n")
    sqlout.write("SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;\n")
    sqlout.write("SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;\n")
    sqlout.write("SET @OLD_SQL_LOG_BIN=@@SQL_LOG_BIN, SQL_LOG_BIN=0;\n")
    sqlout.write("\n")

def _writeEndLoad(sqlout):
    sqlout.write("SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;\n")
    sqlout.write("SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;\n")
    sqlout.write("SET SQL_LOG_BIN=@OLD_SQL_LOG_BIN;\n")
    sqlout.write("\n")
    sqlout.write("SET @ELAPSED_TIME = UNIX_TIMESTAMP(CURRENT_TIMESTAMP())-UNIX_TIMESTAMP(@START_TIME);\n")
    sqlout.write("SELECT CURRENT_TIMESTAMP(), @ELAPSED_TIME;\n")

def _writeStatTable(sqlout):
    sqlout.write("CREATE TABLE IF NOT EXISTS Load_Statistics (\n")
    sqlout.write("  filename      CHAR(20) NOT NULL PRIMARY KEY,\n")
    sqlout.write("  started       TIMESTAMP NOT NULL,\n")
    sqlout.write("  completed     TIMESTAMP NOT NULL,\n")
    sqlout.write("  elapsedTime   INT UNSIGNED NOT NULL,\n")
    sqlout.write("  ipAddresses   INT UNSIGNED,\n")
    sqlout.write("  locations     INT UNSIGNED,\n")
    sqlout.write("  banners       INT UNSIGNED,\n")
    sqlout.write("  ipv4Scans     INT UNSIGNED,\n")
    sqlout.write("  bannerDates   INT UNSIGNED,\n")
    sqlout.write("  html          INT UNSIGNED,\n")
    sqlout.write("  scanHtml      INT UNSIGNED,\n")
    sqlout.write("  properties    INT UNSIGNED,\n")
    sqlout.write("  optProperties  INT UNSIGNED,\n")
    sqlout.write("  sslProperties  INT UNSIGNED,\n")
    sqlout.write("  scanSsl       INT UNSIGNED,\n")
    sqlout.write("  hostDomProps  INT UNSIGNED\n")
    sqlout.write(")ENGINE=MyISAM CHARACTER SET=utf8;\n")

def _writeStats(sqlout,basename):

```

```
    sqlout.write("INSERT INTO Load_Statistics (filename,started,completed,elapsedTime,\n\t\t" + \  
        "ipAddresses, locations, banners, ipv4Scans,bannerDates,\n\t\t" + \  
        "properties,optProperties,sslProperties,scanSsl,HostDomProps)\n\t\t" + \  
        "VALUES ({},@START_TIME,NOW(),ELAPSED_TIME,\n\t\t\t\t" + \  
        "@IPAddresses,@Locations,@Banners,@IPv4_Scans,@Banner_Dates,\n\t\t\t\t" + \  
"@Properties,@Opt_Properties,@SSL_Properties,@Scan_SSL,@Host_Dom_Properties);\n".format(basename))  
  
def _writeUpdateStats(sqlout,basename):  
    sqlout.write("UPDATE Load_Statistics SET html = @HTML, scanHtml = @Scan_HTML " + \  
        "WHERE filename = {};\n".format(basename))
```

REFERENCES

- Afarin, Cyrus. (2017). Can Shodan Keep Up With the Times?. University of Arizona.
- Auffret, P., “SinFP, unification of active and passive operating system fingerprinting,” *Journal in Computer Virology*, vol. 6, no. 3, pp. 197-205, 2010.
- Andreeva, O., Gordeychik, S., Gritsai, G., & Kochetova, O. (2016). Industrial Control Systems Vulnerabilities Statistics, 1–19.
- Arnaert, M., & Antipolis, S. (2016). Modeling Vulnerable Internet of Things on SHODAN and CENSYS : An Ontology for Cyber Security, (c), 299–302.
- Attipoe, Antoinette E.; Yan, Jie; Turner, Claude; Richards, D. (2016). Visualization Tools for Network Security. *Electronic Imaging, Visualization*, 1–8.
- Ayuburi and L. Sobrevinas, “Securing Supervisory Control and Data Acquisition Systems: Factors and Research Direction,” in Americas’ Conference on Information Systems (AMCIS), 2015.
- Bodenheim, R., Butts, J., Dunlap, S., & Mullins, B. (2014). 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.
<http://doi.org/10.1016/j.ijcip.2014.03.001>
- Genge, B., & Enăchescu, C. (2015). Non-Intrusive Historical Assessment of Internet-Facing Services in the Internet of Things, 25–36.
- Guimaraes, V. T., Dal Sasso Freitas, C. M., Sadre, R., Tarouco, L. M., & Granville, L. Z. (2015). A Survey on Information Visualization for Network and Service Management.

Communications Surveys Tutorials, *IEEE*, *PP(99)*, 1.
<http://doi.org/10.1109/COMST.2015.2450538>

Hadlak, S., Schumann, H., & Schulz, H. (2015). A Survey of Multi-faceted Graph Visualization. *Eurographics Conference on Visualization (EuroVis)*, (JANUARY), 1–20.
<http://doi.org/10.2312/eurovisstar.20151109>

Institute, S. (2014). Security Data Visualization. *Worm Propagation and Countermeasures*, 36.

Jeon, S., Yun, J.-H., Choi, S., & Kim, W.-N. (2016). Passive Fingerprinting of SCADA in Critical Infrastructure Network without Deep Packet Inspection, (1). Retrieved from
<http://arxiv.org/abs/1608.07679>

Kehrer, J., & Hauser, H. (2013). Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(3), 495–513. <http://doi.org/10.1109/TVCG.2012.110>

Ko, Y., Ra, I., & Kim, C. (2015). A Study on IP Exposure Notification System for IoT Devices Using IP Search Engine Shodan, 10(12), 61–66.

KOBARA, K. (2016). Cyber Physical Security for Industrial Control Systems and IoT. *IEICE Transactions on Information and Systems*, E99.D(4), 787–795.
<http://doi.org/10.1587/transinf.2015ICI0001>

Langton, J. T., & Newey, B. (2010). <title>Evaluation of current visualization tools for cyber security</title>, 7709, 770910-770910–11. <http://doi.org/10.1117/12.850160>

- Liu, S., Cui, W., Wu, Y., & Liu, M. (2014). A survey on information visualization: recent advances and challenges. *Visual Computer*, 30(12), 1373–1393. <http://doi.org/10.1007/s00371-013-0892-3>
- McKenna, S., Staheli, D., & Meyer, M. (2015). 009 Unlocking user-centered design methods for building cyber security visualizations. *2015 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 1–8. <http://doi.org/10.1109/VIZSEC.2015.7312771>
- Onyeji, M. Brazilian, and C. Bronk, “Cyber Security and Critical Energy Infrastructure,” *Electr. J*, vol 27, no. 2, pp. 52-60, Mar 2014.
- Patton, M., Gross, E., Chinn, R., Forbis, S., Walker, L., & Chen, H. (2014, September). Uninvited connections: a study of vulnerable devices on the internet of things (IoT). In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint* (pp. 232-235). IEEE.
- Rasmeet S. Bali, Neeraj Kumar, Secure clustering for efficient data dissemination in vehicular cyber–physical systems, in *Future Generation Computer Systems*, Volume 56, March 2016, Pages 476-492, ISSN 0167-739X, <http://dx.doi.org/10.1016/j.future.2015.09.004>.
- Riesenfeld, R. F. (2009). A Survey of Radial Methods for Information Visualization, *15*(5), 759–776.
- Rohrmann, Rodney (2017). Large Scale Anonymous Port Scanning. University of Arizona.