Hunting for Malicious Infrastructure Using Big data

Shadi Al-Hakimi
Freek Bax
University of Amsterdam

Jop van der Lelie
Daniel Sierat
NCSC

Content

- Introduction
- Problem statement
- AI and Machine learning
- Implementation/Data gathering
- Results/discussion
- Conclusion
- Future work

Introduction

- Malicious infrastructure : Malicious Command and Control Web Servers such as (Cobalt strike).
- They try to hide there messages in HTTP because it is not blocked by firewalls and it is hard to analysis all HTTP traffic.
- Fingerprinting is one of the techniques used to identify malicious C&C servers.
- Most research on anomaly detection focused on outgoing and incoming HTTP traffic of a specific host or network. (needs active beacon)
- Passive fingerprinting of HTTP GET root (/) responses could identify active C&C servers before beacon deployment.

Problem statement

How can machine learning be used on a data set of HTTP responses to identify malicious webservers?

- What features can we extract from the HTTP responses? (Empirical Approach)
- Which machine learning algorithm are best suited for anomaly detection on a data set of HTTP responses? (Theoretical analysis)

AI and Machine learning

- Supervised versus unsupervised learning
- Creating a labeled data set
- Clustering algorithms

Feature extraction

- Transforming text
- Natural language processing

- **Header ordering**

```
Server: Varnish
Retry-After: 0
content-type: text/html
Cache-Control: private, no-cache
X-Served-By: cache-pao17442-PAO
Content-Length: 247
Accept-Ranges: bytes
Date: Mon, 11 Jan 2021 05:03:15 GMT
Via: 1.1 varnish
Connection: close
```

Uniqueness

- Counting the frequency of all header fields
- Computing the uniqueness for each header field
- Uniqueness of a header = 1 - frequency header / total responses

- For each HTTP response computing their uniqueness
- Adding the uniqueness value of all the headers present
- Adding 1 - uniqueness for all headers absent

Evaluation

- Looking at the distribution of the features
- Finding anomalies with the features

Implementation

- Dataset: Part of project sonar from Rapid7 specifically IPv4 space scan of port 80 HTTP GET root (/).
- Technologies: Dask a python library for parallel processing of large-scale datasets.
- Hardware: Group (cluster) of 3 machines (160GiB RAM, 60 core CPU)
- Data processing:
  - Preprocessing.
  - Dask processing.
- Uniqueness Algorithm
- Header order Algorithm

Implementation: Data processing ... con’t 2

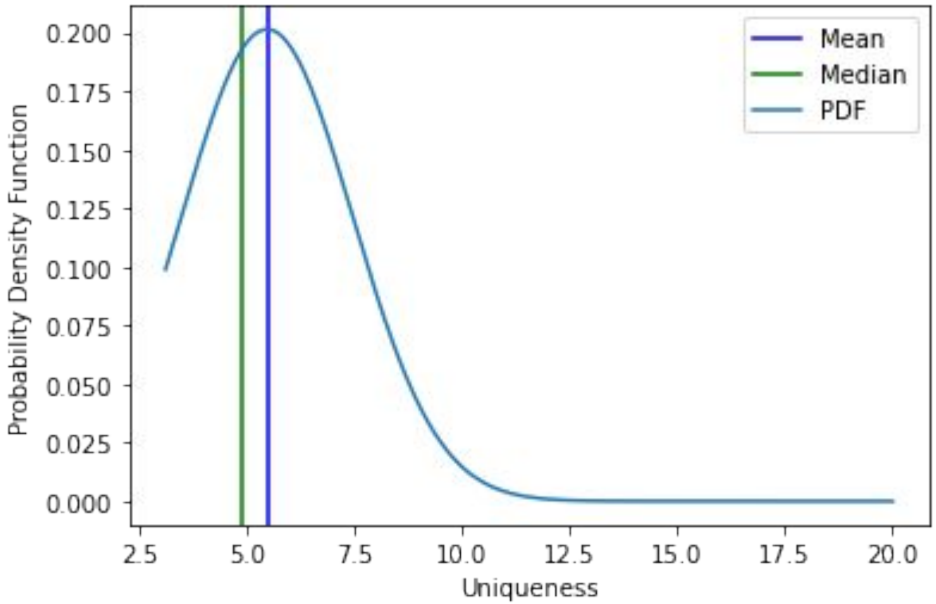
- Preprocessing
  - Each raw of the original dataset contain the ip and the base64 of the response.
  - We wrote a python script to decode the base64 encoding into json object {'ip':’<IPv4adress>’, ‘headers':’<HTTPHeaders>’}
- Dask processing
  - We changed data file format from json to parquet.
  - Loaded the data into Dask.
  - Proper partitioning of the data is really important!.
  - Started to run queries. `df1 = df['headers'].str.extractall('(\n[\w\-]*\:|^[\w\-]*:)')`

Implementation: feature extraction ... con’t 3

```
## Uniqueness Algorithm

For each response in dataset:

    For each header_name in uniqueness_values:

        If header_name inside response:

            Add uniqueness_value[header_name]

        Else:

            Add (1.0 - uniqueness_value[header_name])

  For each header_name in response:

      If header_name not in uniqueness_values:

          Add 0.99
```

```
## Header order Algorithm

For each response in dataset:

    row = getIndex.header_names() # [160,12,30,190,0]

    row= row.sort_values() # [0,12,30,160,190]

    For value in row.index:

        If index > -1:   # header_name not exists

            result.append(value)

    response[‘order] = result   # [2,1,3,4,5]
```

Results


Mean
Median
PDF
Probability Density Function
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Uniqueness

Ordering

- Not distributed evenly
- Majority of orderings uncommon

% the data set	Number of unique orderings
50%	4
75%	10
90%	32
95%	55
99%	87
99.9%	168
99.99%	230
100%	301

Ordering

Ordering	Frequency
Server, Content-type, Content-length, Date, Connection	9824687
Server, Date, Content-length, Content-type, Connection	8099211
Date, Server, Content-length, Connection, Content-type	7875328
Server, Date, Content-type, Connection	4528994
Content-type, Server, Date, Connection, Content-length	3612256
Date, Content-type, Content-length, Connection, Server	1469608
Date, Server, Content-length, Content-type, Connection	1239995
Date, Server, Connection, Content-type	1041208
Date, Server, Connection, Content-length Content-type	980881
Date, Content-type, Content-length, Connection	571911

Ordering

	Connection	Date	Content-type	Server	Content-length	Average
Connection		4.62%	25.49%	7.68%	17.97%	13.93739275%
Date	95.38%		61.66%	35.74%	67.55%	65.08130996%
Content-type	74.51%	38.34%		18.13%	70.49%	50.36916664%
Server	92.32%	64.26%	81.87%		91.09%	82.38651886%
Content-length	82.03%	32.45%	29.51%	8.90%		38.22561181%

Conclusion

- Both uniqueness and ordering feature are promising
- Clustering looks best suited to use on HTTP responses

Future research

- Using the features with machine learning
- Comparing different clustering algorithms
- Other features

Questions?

https://gitlab.com/shadialhakimi/rp1

Thank you