

# Ransomware Detection using Supervised Learning

Ali Bhangoo, Alan Tran, Peter Lam, Supervisor: Miguel Garzón  
April 21st, 2020

## **Table of Contents**

<b>How COVID-19 is Affecting Ransomware</b>	<b>2</b>
<b>What is Emotet?</b>	<b>2</b>
<b>Project Goals</b>	<b>4</b>
<b>Data Collection</b>	<b>5</b>
Limitations and Roadblocks	7
Automation	9
<b>Data Visualization</b>	<b>10</b>
<b>Emotet and Trickbot Detection Using One-Class SVM</b>	<b>12</b>
Dataset Collection	12
Layout	14
Using Docker or Running Locally Off the Box	16
Pipeline	17
Training With New Data	19
Validate With New Data	19
Pre-processing	19
Feature Generation	20
Training	21
Testing Metrics	22
Validation Metrics	22
Confusion Matrix	23
<b>Future work</b>	<b>24</b>
<b>References</b>	<b>25</b>

## How COVID-19 is Affecting Ransomware

Ransomware limits users from accessing their system by encrypting important files. Criminals demand their victims to pay a ransom, usually with cryptocurrency to regain control of their system and data.

The COVID-19 pandemic has made organizations like hospitals, universities, and governments, more conscious about losing access to their systems and more motivated to pay the ransom. Cyber criminals take advantage of this situation by running faster and more ransomware attacks, and offering ransomware-as-a-service. COVID-19 is being used in a variety of malicious campaigns including email spam, malware, ransomware, and malicious domains.

## What is Emotet?

Emotet is a Trojan that is most commonly spread by email, using malicious script, macro-enabled document files, or malicious URLs. The malware was first detected in 2014, and remains active, deemed one of the most prevalent threats of 2019. The original versions of the Emotet malware functioned as a banking trojan aimed at stealing credentials from victims [1]. Throughout 2016 and 2017, the operators of Emotet updated the trojan to primarily be used as a “loader”, a type of malware used to gain access to a system and then download additional payloads. These second-stage payloads can take various forms, ranging from executable code from Emotet’s own modules to malware developed by other operators.

Through the many iterations of Emotet, it finally evolved to use macro-enabled documents to retrieve the virus payload from command and control (C&C) servers which are run

by the attackers. Compromised systems often contact Emotet's C&C servers to retrieve updates and new payloads [1]. This allows the attackers to install additional malware such as other bank Trojans, and install updated versions of the software.

The primary distribution for Emotet is through malspam. Emotet gains access to your contacts list and sends itself to everyone and since these emails are coming from your infected email account, the emails look trustworthy to your contacts, making them more inclined to click bad URLs and download infected files. If there is a connected network on the victims device, Emotet spreads using a list of common passwords, brute-forcing its way onto other connected systems. As a result, the Emotet botnet is very active and responsible for a large amount of malspam we encounter [1]. Once infected, computers are added to the Emotet botnet. In addition, Emotet is known for renting access to infected computers to ransomware operations, such as TrickBot and Ryuk. The Emotet gang is also known to run their botnet as a Malware-as-a-Service (MaaS) which allows other cybercriminal gangs to rent access to Emotet-infected computers and drop their own malware strains alongside Emotet.

The notorious malware Trickbot has been linked to more COVID-19 phishing emails than any other, per new data from Microsoft [2]. As we know, Emotet is a dropper for the TrickBot trojan, and then TrickBot steals sensitive information and downloads the Ryuk ransomware. Figure 1 below demonstrates the steps in which the attack occurs.

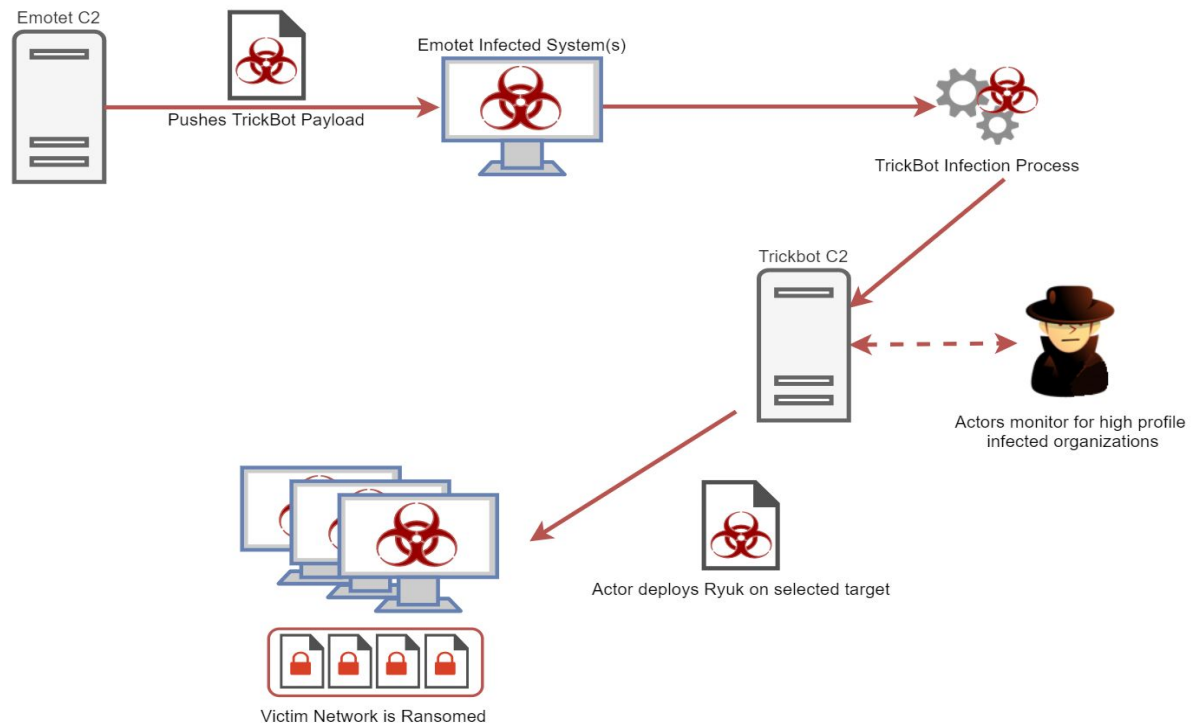


Figure 1: Flow of the attack as Emotet delivers TrickBot, which delivers Ryuk in [3]

As mentioned above, there has been an increase in malware and ransomware attacks due to the COVID-19 pandemic. Both Microsoft and Google made claims many of the COVID-19 themed threats are not new, but simply rebranded with coronavirus themes, such as TrickBot [2].

## Project Goals

As a user, detecting malicious email and websites is becoming increasingly hard. Malicious attackers now use social engineering to make emails more personal and believable, such is the case in Emotet where the email comes from a trustworthy person. Due to the COVID-19 it is more likely than ever that coworkers, friends, and family will accidentally download or open a malicious file or click a malicious link. There is a great chance that it might

not be detected by antivirus software as malicious programs are constantly evolving to evade antivirus measures. These detection softwares cost a lot of money and have privacy issues which deter users from purchasing and installing them. With all these factors in mind, our project goal was to create a tool using machine learning so that we can detect malicious ransomwares like Emotet. We hope to use machine learning to continuously identify new evolutions of the malware.

## Data Collection

Once the initial spiking phase was completed, our team began to collect various indicators of compromise (IoC). These indicators included compromised IPs, domains, URLs, and hashes through sources such as PasteBin, Twitter, Malwarebytes, and VirusTotal.



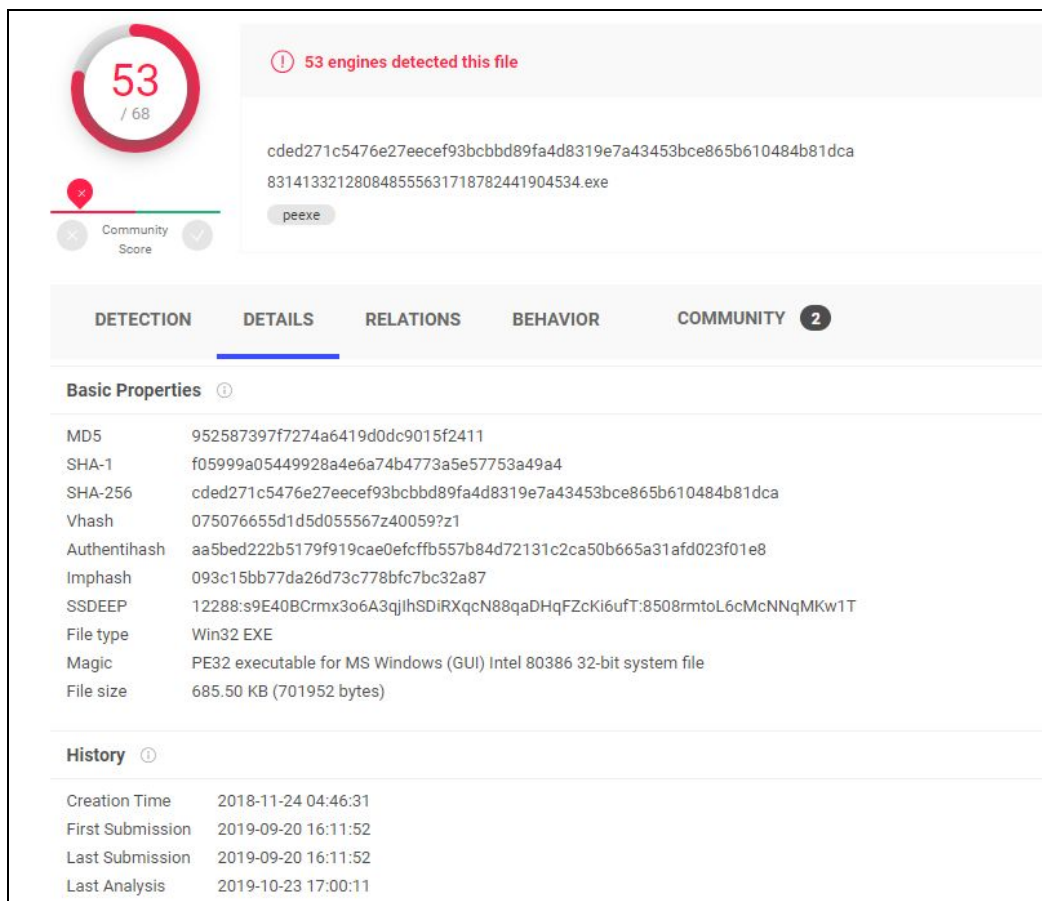
Figure 2: Ryuk IoCs from Malwarebytes blog in [4]

Initially, our project scope was focused on detecting Ryuk, another ransomware, however due to its infancy, our team was unable to gather a large dataset for it. Instead, our team decided

to increase our scope to include Trickbot and Emotet. With the addition of other ransomwares, the goal of data collection was to serve to main purposes:

- Utilize Emotet hashes for machine learning
- Visualize and track trends from IPs, domains, URLs, and various categories of ransomware

After the initial data was collected, each IoC needed to be verified and analyzed for the confidence rate. Due to the large amount of data, an automated process would need to be implemented. Using VirusTotal's API, an online tool for analyzing suspicious files and URLs, our team was able to gather information about each IoC.



The screenshot displays the VirusTotal analysis interface for a file. At the top left, a circular progress indicator shows 53 out of 68 engines detected the file as malicious. Below this, a 'Community Score' bar is visible. The main header area contains a warning icon and the text '53 engines detected this file'. The file's SHA-256 hash is displayed as 'cded271c5476e27eecef93bcbbd89fa4d8319e7a43453bce865b610484b81dca', and the file name is 'peexe'. The interface includes tabs for 'DETECTION', 'DETAILS', 'RELATIONS', 'BEHAVIOR', and 'COMMUNITY' (which has a notification badge for 2 items). The 'DETAILS' tab is currently selected, showing 'Basic Properties' and 'History' sections.

Basic Properties	
MD5	952587397f7274a6419d0dc9015f2411
SHA-1	f05999a05449928a4e6a74b4773a5e57753a49a4
SHA-256	cded271c5476e27eecef93bcbbd89fa4d8319e7a43453bce865b610484b81dca
Vhash	075076655d1d5d055567z40059?z1
Authentihash	aa5bed222b5179f919cae0efcfeb557b84d72131c2ca50b665a31afd023f01e8
Imphash	093c15bb77da26d73c778bfc7bc32a87
SSDEEP	12288:s9E40BCrmx3o6A3qjlhSDiRXqcN88qaDHqFZcKi6ufT:8508rmtol6cMcNNqMKw1T
File type	Win32 EXE
Magic	PE32 executable for MS Windows (GUI) Intel 80386 32-bit system file
File size	685.50 KB (701952 bytes)

History	
Creation Time	2018-11-24 04:46:31
First Submission	2019-09-20 16:11:52
Last Submission	2019-09-20 16:11:52
Last Analysis	2019-10-23 17:00:11

Figure 3: Example of information gained through VirusTotal in [5]

## Limitations and Roadblocks

VirusTotal, like any other business, requires users to pay for a private API key for in depth and additional features. Some features include unlimited API calls, network traffic information, behavior information, and the ability to download virus samples. Unfortunately due to limitations within our team, we were unable to utilize VirusTotal as initially intended. By using the public API key instead, our team encountered the following problems:

- A request rate limitation of 4 requests/minute
- Daily quota limit of 1000 requests/day
- Inability to download network traffic for machine learning

Due to the request limitations, our team realized we would not have the capabilities of scanning thousands of IoCs in a realistic and reliable way. Alternatively, we discovered that rotating through several dozen API keys allowed our team to bypass the throttling of API calls, thus reducing the run time from several days to only 20 minutes. By utilizing this method, we were able to collect basic information about an IoCs confidence rate and scan date. Due to the lack of information available, we also utilized various other APIs such as IPInfo [6] and Cymruwhois [7], to gather information such as the longitude, latitude, region, city, postal code, organization name, and ASN information of a given IoC. In addition, we also parsed the information from domains and URLs to track common file names, extensions, hostnames and queries.

```

"type": "DOMAIN",
"value": "efreedommaker[.]com",
"malware": "emotet",
"source": "https://www.cybereason.com/blog/triple-threat-emotet-deploys-trickbot-to-steal-data-spread-ryuk-ransomware",
"source_url_query": "",
"source_url_path": "/blog/triple-threat-emotet-deploys-trickbot-to-steal-data-spread-ryuk-ransomware",
"source_url_filename": "",
"source_url_file_ext": "",
"source_url_scheme": "https",
"source_url_hostname": "www.cybereason.com",
"source_url_domain": "www.cybereason.com",
"is_valid": 1,
"total_detected_urls": 35,
"total": 2388,
"positives": 157,
"percent_score": 6.57,
"latest_scan_date": "2020-01-17 08:22:05",
"first_scan_date": "2018-02-12 21:38:42",
"city": "Ann Arbor",
"latitude": 42.2535,
"longitude": -83.8366,
"postal": "48106",
"region": "Michigan",
"org": "AS55293 A2 Hosting, Inc.",
"ip": "162.212.130.105",
"asn": "55293",
"is_bell": false,
"value_url_query": "",
"value_url_path": "",
"value_url_filename": "",
"value_url_file_ext": "",
"value_url_scheme": null,
"value_url_hostname": "efreedommaker.com",
"value_url_domain": "efreedommaker.com"

```

Figure 4: Information gained from a single website URL in [8]

Unfortunately, the API keys made available to our team were not capable of downloading the virus samples and network traffic data from the hashes that were collected. Since this was a crucial part of the machine learning portion of the project, an alternative would need to be found. As a result, we discovered that Stratosphere Research Laboratory, a project created by the Czech Technical University, had publically available network traffic datasets of Emotet and Trickbot [9]. Using the collected dataset from this resource, our team was able to proceed with the pre-processing, feature generation, and training of our model. Since the dataset size was smaller than anticipated, our team continued to collect hashes from various other sources to ensure that our work could be expanded on in the future through VirusTotal or other similar resources.



## Automation

The list of compromised IPs, domains, URLs, and hashes are always changing. While IoCs can be quickly removed by service providers, new IoCs are still rapidly being created. With an always evolving field, the data sets need to be easily interchanged with updated and relevant data sets. Although the initial data collected by our team is acceptable for our project scope, our goal was to make data collection easier for any future work.

In doing so, we developed several python scripts to help reduce the overhead needed to collect IoC information for data visualization and machine learning in the future. During our initial data collection phase, we found that sources tend to post hundreds of IoCs at a time. Since these lists are sometimes crowd sourced or posted anonymously, verifying these IoCs was very important.

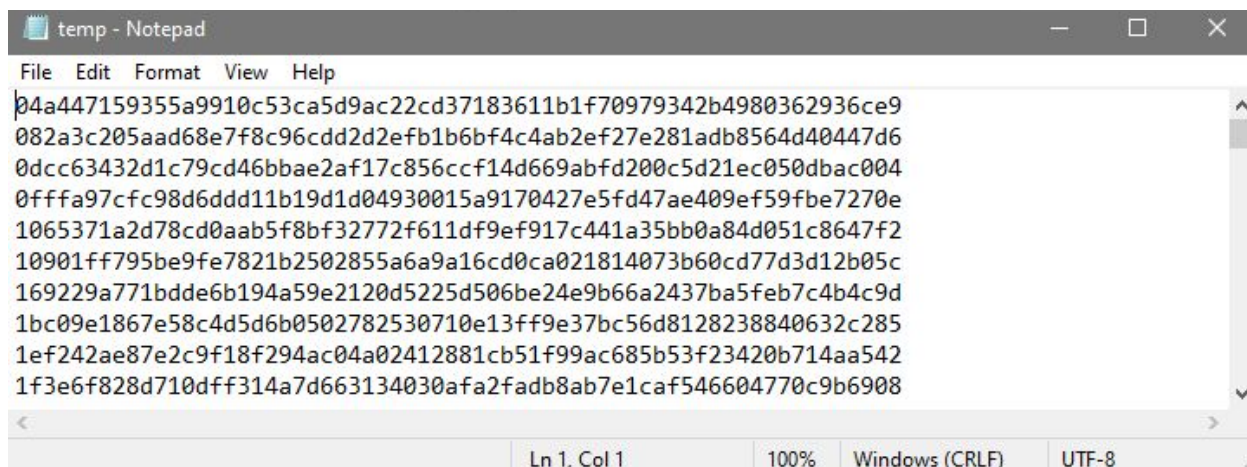


Figure 5: Sample input of a list of Emotet hashes

```

7114 {"type": "URL", "value": "http://elarabia.designlandwebsolutions.online/wp-admin/cojve06f-0p5b4-4656/", "malware": "emotet", "source": "https://paste.cryptolaemus.com/"}
7115 {"index": {"_index": "ioc", "_id": 3557}}
7116 {"type": "URL", "value": "http://txblog.50cms.com/wp-admin/10yg3j3l-pggp7p80-519/", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/2020/02/07/0/"}
7117 {"index": {"_index": "ioc", "_id": 3558}}
7118 {"type": "URL", "value": "http://marahiyohiyo.com/wp-admin/xwTa5d/", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/2020/02/07/09-emotet-malware-"}
7119 {"index": {"_index": "ioc", "_id": 3559}}
7120 {"type": "URL", "value": "http://wqapp.50cms.com/addons/xrxUPWg/", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/2020/02/07/09-emotet-malware-"}
7121 {"index": {"_index": "ioc", "_id": 3560}}
7122 {"type": "SHA256", "value": "04a447159355a9910c53ca5d9ac22cd37183611b1f70979342b4980362936ce9", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}
7123 {"index": {"_index": "ioc", "_id": 3561}}
7124 {"type": "SHA256", "value": "082a3c205aad68e7f8c96cdd2d2efb1b6bf4c4ab2ef27e281adb8564d40447d6", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}
7125 {"index": {"_index": "ioc", "_id": 3562}}
7126 {"type": "SHA256", "value": "0dcc63432d1c79cd46bbae2af17c856ccf14d669abfd208c5d21ec058db4004", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}
7127 {"index": {"_index": "ioc", "_id": 3563}}
7128 {"type": "SHA256", "value": "0fffa97cfc98d6ddd11b19d1d04930015a9170427e5fd47ae409ef59f67270e", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}
7129 {"index": {"_index": "ioc", "_id": 3564}}
7130 {"type": "SHA256", "value": "1065371a2d78cd0aab5f8bf32772f611df9ef917c441a35bb0a84d051c8647f2", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}
7131 {"index": {"_index": "ioc", "_id": 3565}}
7132 {"type": "SHA256", "value": "10901ff795be9fe7821b2502855a6a9a16cd0ca021814073b60cd77d3d12b05c", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}
7133 {"index": {"_index": "ioc", "_id": 3566}}
7134 {"type": "SHA256", "value": "169229a771bde6b194a59e2120d5225d506be24e9b66a2437ba5feb7c4b4c9d", "malware": "emotet", "source": "https://paste.cryptolaemus.com/emotet/"}

```

Figure 6: Sample file output in [8]

By providing lists of IoCs, users are able to utilize *bulk\_json\_generator.py* to receive meaningful information about a given IoC's confidence rate, location of origin, ASN, and more [8]. In addition, the produced file allows for tools like ElasticSearch to read and extract information for data visualization tools like Kibana.

## Data Visualization

Using Kibana, our team was able to visualize our collected dataset to look for possible trends and correlations. In addition, this allowed our team to receive a high level view of the quality and the diversity of our data. The following is a Kibana dashboard that represents the collection of 6832 different IoCs associated with Emotet, Trickbot, and Ryuk.



Figure 7: Kibana dashboard showing overview information in [8]

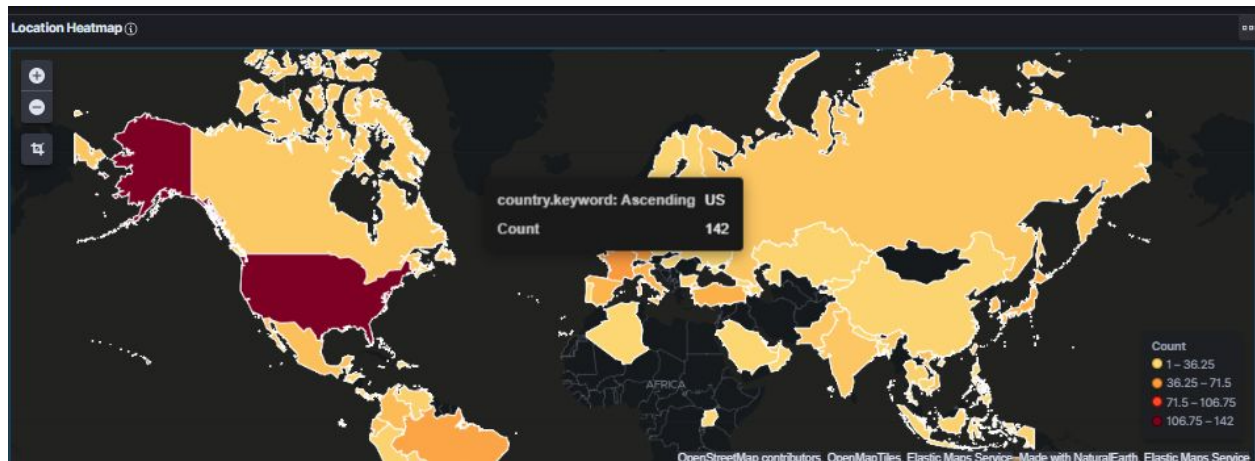


Figure 8: Heatmap of the locations of registered IPs, domains, and URLs in [8]



Figure 9: Top ASN organizations used by reported Emotet IoCs in [8]

Since we were able to extract ASN information from given IoCs, we were able to look at common organizations that certain ransomwares were using. Notably, DigitalOcean was used most commonly by Emotet, while Trickbot IoCs favoured the use of OVH. Given that both organizations have a publically available resource to report abuse, our information collected could easily be used to alert various organizations of possible malicious sites.

Although the dataset visualized in this example was not used in our machine learning process, we can still use the information collected to reactively alert organizations of malicious content. In the future, once the network flows of the collected hashes can be collected through VirusTotal, the dashboard will allow our team to filter our dataset to meet specific requirements for our training model.

## **Emotet and Trickbot Detection Using One-Class SVM**

Here we will go through the major parts of our data collection, preprocessing, feature building, training, validation, and as well as discuss the results.

### **Dataset Collection**

As mentioned before, the source for our training set is retrieved from Stratosphere Research Laboratory. A script called `src/data/scripts/binetflow_collection.py` takes an input file `src/data/inputs/dataset_url_list.json` which contains the URLs to download the binetflows and what host ips in each binetflow are malicious. This input file is created manually and can be modified to add/delete the binetflow you want to exclude from training. Binetflows can be read as a csv. When the files are downloaded, they are saved as csv file types.

```
[
  {
    "source":
    "https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-114-1/2015-04-09_capture-win2.binetflow",
    "malware": "emotet",
    "type": "MD5",
    "infected_host": ["10.0.2.102"],
    "infected_gw": "",
    "value": "8baa9b809b591a11af423824f4d9726a"
  }
]
```

Figure 10: Sample JSON

Viewing the file can show all file names of all malicious captures being used for training.

Field	Description
StartTime	Record start time
Dur	Duration of a flow
Proto	Transaction protocol
SrcAddr	Source IP address
Sport	Source port
Dir	Direction of transaction
DstAddr	Destination IP address
Dport	Destination port
State	Transaction State
sTos	Source TOS byte value
dTos	Destination TOS byte value
TotPkts	Total transaction packet count
TotBytes	Total transaction bytes
SrcBytes	Bytes sent from source to destination
SrcUdata	Data sent from source to destination
dstUdata	Data sent from destination to host
Label	Label of each flow row (1 infected, benign 0)

Figure 11: Each column and their description in [10]

## Layout

Our data has been laid out in between many sub-folders. Each sub-folder contains csv's that are the result of running scripts through our pipeline. Here's the order each sub-folder would appear from executing the scripts:

1. External - contains the initial malicious data retrieved from Stratosphere Research Laboratory. A script is run to send requests to gather the bidirectional network flow of computers and networks infected with emotet and trickbot. This data is already in the GitHub repository as there are times when the domain is unresponsive.
  - 2 sub-folders, binetflow and validation. Binetflow contains all csvs for training whereas validation is all csv used for validation.
2. Raw - contains a file called input.csv and is the concatenation of all csvs from external/binetflow. Each row labeled 1 or 0 if the Src or Dst addresses are infected which is determined by dataset\_url\_list.json.
3. Interim - takes the input.csv, removes the columns srcUdata and dstUdata, and removes all rows that do not have an IPv4 Src or Dst address or have empty addresses or ports.
4. Preprocessed - uses the interim.csv and preprocesses the csv by adding new columns such as binning of Sport and Dport, and mapping the Dir, State, and Protocols into integers. It also adds a new column is\_fwd.
5. Processed - contains a file called processed.csv which uses the preprocessed.csv and builds features on top.
6. Trained - is a new CSV called trained.csv which is generated after training with processed.csv and adding new columns for predicted results and confidence scores.
7. Validation - similar output as trained.csv but with validation data.

\* **Note:** validation data goes through the same exact procedures but you must switch the config file if you want to generate a processed validation csv. This file will be put in the GitHub repository for convenience.

```

|—— data
|   |—— external      <- Data from third party sources.
|   |   |—— binetflow <- Bidirectional netflow files for training set.
|   |   |—— validation <- Bidirectional netflow files for validation.
|   |—— interim      <- Intermediate data that has been transformed.
```





## Using Docker or Running Locally Off the Box

### Docker

Dockerfiles are given to run `predict_model` or `train_model`. These can be found under folders `DockerPredict` and `DockerTrain`. A prerequisite would be to install Docker. If on a Windows laptop and using Docker within a VM, I would recommend a computer with more than 16GB of RAM. Currently, a beta version for Docker Windows is available but requires an insider beta Windows 10 program and the current latest beta version of Windows 10. The other option is to have a Linux machine, Linux VM, or MacOS. Since the data size is more than a million, it is memory taxing and if you receive container exit status 137, then the machine does not have enough memory. If the containers do not work, a better solution will be to try locally.

### Example: Build and Run a Container

**Make sure `processed.csv` and `val_processed.csv` are uncompressed beforehand! Read the repository to see how.**

```
$ docker build -f DockerPredict/Dockerfile -t predict_model:latest .
$ docker run -d predict_model:latest
```

### Monitor and interact with container:

```
$ docker ps -a
$ docker exec -it (container id) bash
```

### Viewing results:

- The result of predicting will be under `RDM/data/validation/validation.csv`
  - The result of training will be under `RDM/data/trained/{model}.csv`
  - All metrics can be found in `RDM/metric`
-



**Locally:** Running locally is very similar. Create a python3 venv or install the requirements in the default bin. **Make sure processed.csv and val\_processed.csv are uncompressed beforehand!**

```
$ pip install -r requirements.txt
$ cd ./src/models/
$ python predict_model.py
```

Or

```
$ python train_model.py
```

### Viewing results:

- The result of predicting will be under  
Ransomware-Detection-Mechanism/data/validation/validation.csv
- The result of training will be under  
Ransomware-Detection-Mechanism/data/trained/{model}.csv
- All metrics can be found in Ransomware-Detection-Mechanism/metric

## Pipeline

This shows the pipeline of our machine learning. Here you can see the timings of each step.

```
├── 1. Preparing Data (1.3 mil rows) <- /src/data> python
make_dataset.py
|   ├── a. Download data sets      (8.5 min)
|   ├── b. Create raw data        (8.5 sec)
|   ├── c. Create interim data    (8 sec)
|   └── d. Create preprocessed data (30 sec)
├── 2. Build Features (1.3 mil rows) (2.28 hours) <- /src/features>
python build_features.py
├── 3. Train Model (1.3 mil rows) (6.38 hours) <- /src/models>
python train_model.py
|   ├── One Class SVM              (5.03 hours)
|   ├── Confidence SCore           (36.4 min)
|   ├── Save OC Features CSV       (4.4 min)
|   ├── Linear Regression          (22.2 min)
|   └── Save LR Features CSV       (4.6 min)
└── 4. Predict Model (24.7 min) <- /src/models> python
predict_model.py
```

1. Preparing Data - Handled by `/src/data/make_dataset.py`: creates `preprocessed.csv` given the external data of our dataset. `Make_dataset.py` calls sub-scripts at the location `/src/data/scripts/`
  - a. Download data sets - `binetflow_collection.py`: downloads the binetflows if the user does not already have it.
  - b. Create raw data - `raw_data.py` creates `input.csv`
  - c. Create interim data - `interim.py` creates `interim.csv`
  - d. Create preprocessed data - `preprocess.py` creates `preprocessed.csv`
2. Build Features - Handled by `/src/features/build_features.py`. Generate features and create `processed.csv`.
3. Train Model - `/src/models/train_model.py`: trains a model using OC SVM. Tests the newly made model with test data before calculating the confidence and training using LR, while saving each model made and metrics such as accuracy, confusion matrix, etc.
4. Predict Model - `/src/models/predict_model.py`: validates model using processed validate data. This validation data is already generated before-hand and should be in the repository. You may need to download externally. See repository.

**To run everything by scratch (besides creating validation data) follow these steps:**

```
$ cd project/src/data
$ python make_dataset.py
$ cd ../features
$ python build_features.py
$ cd ../models
$ python train_model.py
$ python predict_model.py
```

## Training With New Data

If you would like to use new data for training or edit which captures are used, edit the `project/src/data/inputs/dataset_url_list.json`. Afterwards, run the pipeline as seen above!

## Validate With New Data

This is a similar process but with extra steps.

1. Edit `project/src/data/inputs/validation_url_list.json`

2. Edit scripts to use the validation config file
  - Each part of the pipeline besides models (data, features) will have two YAML config files. By default, the scripts use the config.yml and features\_config.yml respectively in the pipeline.
    - Edit these files such that:
      - Data
        - All scripts under project/src/data/scripts/\* have CONFIG\_PATH = validation\_config.yml
      - Features
        - build\_features.py CONFIG\_PATH = val\_features.config.yml
3. Run make\_data.py and build\_features.py to generate val\_processed.csv which is used for validation in predict\_model.py

## Pre-processing

Preprocessing is done by binning the source and destination port columns along with mapping the Dir, State, and Proto columns into integers. Lastly, a new column called is\_fwd is made which is a boolean whether the flow is in the forward direction or backward direction.

### How are the bins decided for ports?

```
(0 <= port <= 1023) = 0
(1024 <= port <= 49151) = 1
(49152 <= port <= 65535) = 2
```

### How are the mappings to integers decided?

```
Dir: each unique is assigned an arbitrary and unique integer.
State: each unique is assigned an arbitrary and unique integer.
Proto: most values correspond to iana protocol numbers [11].
Is_fwd = 1 if source port is >=1024 else 0
```

## Feature Generation

Features were generated based on three main columns: SrcBytes, TotBytes, TotPkts. But we needed to calculate the time between two flows as well. Here are the list of the main features:

- Total flows in the **forward** direction in the **window**
- Total flows in the **backward** direction in the **window**
- Total size of netflows in **forward** direction in the **window**
- Total size of netflows in **backward** direction in the **window**
- Minimum size of flow in **forward** direction in the **window**
- Minimum size of flow in **backward** direction in the **window**
- Maximum size of flow in **forward** direction in the **window**
- Maximum size of flow in **backward** direction in the **window**
- Mean size of flow in **forward** direction in the **window**
- Mean size of flow in **backward** direction in the **window**
- Standard Deviation size of flow in **forward** direction in the **window**
- Standard Deviation size of flow in **backward** direction in the **window**
- Time between 2 flows in the window in the **forward** direction
- Time between 2 flows in the window in the **backward** direction

This list is applied to two windows for all three columns. One window is within 10 minutes. The other is within 10k flows. This is for general features. To be more specific the list and both windows are applied with respect to the current source IP and destination IP. This gave a total of 169 features created. One other feature that is added is the DstBytes. A naming convention is made to read the columns. Ex. STotalFlowFwd\_10T and DSrcBytesMaxBwd\_10000N

The first letter is if each row is with respect to the source or destination IPs. It is then followed by the column or feature name if no column. Afterwards, the aggregate function (max, min, std, mean, sum) and the direction (Fwd, Bwd) are appended to the name. Lastly, the window is the suffix of the column name. 10T represents 10 minutes whereas 10000N represents 10k flows as the window.

## Training

The two classifiers used are One-Class SVM and Logistic Regression. LR is used to calibrate the confidence score in OC. Training is done by using a very popular Python library called Scikit-learn. To find what parameters to use for each classifier, we grid search cross validation for OC and randomized search cross validation for LR. The parameters for each classifier:

### One-Class SVM

- $\text{Nuu} = 0.01$
- $\text{Gamma} = 1 * 10^{-6}$

### Logistic Regression

- $C = 69.54618247583652$
- $\text{Tol} = 0.0009555227427965779$
- $\text{Solver} = \text{saga}$
- $\text{Penalty} = \text{l2}$
- $\text{Max\_iter} = 80000$
- $\text{Dual} = \text{False}$

OC SVM was chosen due to the huge class-imbalance between malicious data and benign data. We had more malicious data than benign. Only 13% of the whole dataset is benign. Since OC SVM only trains with one-class, we separated the data between malicious and benign. The testing data consisted of all benign data plus 20% of the malicious data.

LR training set is all data with additional columns of the predicted labels from OC and OC confidence score which is the decision function when classifying rows. Let's look at the testing and validation metrics.

## Testing Metrics

Classifier	Accuracy	Recall	Precision	F1	Average Precision
One-Class SVM	71%	99%	67%	80%	67%
Logistic Regression	N/A	N/A	N/A	N/A	N/A

Figure 12: Testing metrics for One-Class SVM and Logistic Regression

## Validation Metrics

Classifier	Accuracy	Recall	Precision	F1	Average Precision
One-Class SVM	89%	90%	97%	94%	97%
Logistic Regression	90%	95%	94%	95%	94%

Figure 13: Validation metrics for One-Class SVM and Logistic Regression

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{F1} = 2 \times \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 14: Equations of precision, recall, and f1 metrics in [12]

$$\text{Accuracy} = \frac{\text{Number of Correct predictions}}{\text{Total number of predictions made}}$$

Figure 15: Equations of the accuracy metric in [13]

$$\text{AP} = \sum_n (R_n - R_{n-1}) P_n$$

Figure 16: Average precision metric formula “where  $P_n$  and  $R_n$  are the precision and recall at the  $n$ th threshold” in [14]

## Confusion Matrix

### Testing One Class

Predicted	-1	1
Actual		
-1	32%	68%
1	1%	99%

### Validation One Class

Predicted	-1	1
Actual		
-1	74%	26%
1	9%	91%

### Validation Logistic Regression

Predicted	-1	1
Actual		
-1	37%	63%
1	5%	95%

## Future work

As the work of our team begins to wrap up following the end of our semester, there are still various improvements and features that could be implemented in the future. In regards to data visualization, populating the Kibana dashboards with a more diverse data set could yield more interesting trends such as the behavioral differences between ransomwares. The addition of a strong dataset could also benefit in the cluster analysis of epochs.

In terms of the machine learning portion, the LR will need to be recalibrated to clearly reflect OC scores. Currently the LR is trained independently with all the data plus the decision function from the OC but the predictions differ from the one-class model. To improve results further, clustering can be done beforehand to remove noise and remove outliers. If we were to repeat the project again, we would recommend not using Stratosphere Research Laboratory's own .binetflow files but instead use their PCAPs and use CICFlowMeter to convert the PCAP into a binetflow. This is because the formats between the two are different one so we cannot use CICFlowMeter to generate new data to use. It is difficult to find .binetflows in the same format as Stratosphere Research Laboratory and so you're unable to validate with any network flow.

To conclude, ransomware has become an increasingly growing problem as the viruses evolve to being more malicious and harder to detect. This can even be seen through the COVID-19 Pandemic. The automation and data visualization provides a reactive approach to take action when an IP is compromised. The machine learning aspect provides a proactive approach that can be used as an insight on if a user has been infected with emotet. Hopefully our project can give any insight on improving our community.



## References

- [1] “Emotet Malware – An Introduction to the Banking Trojan,” *Malwarebytes*. [Online]. Available: <https://www.malwarebytes.com/emotet/>. [Accessed: 21-Apr-2020].
- [2] P. Muncaster, “Trickbot Named Most Prolific #COVID19 Malware,” *Infosecurity Magazine*, 20-Apr-2020. [Online]. Available: <https://www.infosecurity-magazine.com/news/trickbot-named-most-prolific/>. [Accessed: 21-Apr-2020].
- [3] “North Korean APT(?) and recent Ryuk Ransomware attacks,” *Kryptos Logic*. [Online]. Available: <https://www.kryptoslogic.com/blog/2019/01/north-korean-apt-and-recent-ryuk-ransomware-attacks/>. [Accessed: 21-Apr-2020].
- [4] Umawing and J. Umawing, “Threat spotlight: the curious case of Ryuk ransomware,” *Malwarebytes Labs*, 20-Dec-2019. [Online]. Available: <https://blog.malwarebytes.com/threat-spotlight/2019/12/threat-spotlight-the-curious-case-of-ryuk-ransomware/>. [Accessed: 21-Apr-2020]
- [5] *VirusTotal*. [Online]. Available: <https://www.virustotal.com/gui/home>. [Accessed: 21-Apr-2020].
- [6] Geotek, “IP Address Info,” *IP Address Info*. [Online]. Available: <http://ipinfo.info/>. [Accessed: 21-Apr-2020].
- [7] “API¶,” *API - cymruwhois v1.0 documentation*. [Online]. Available: <https://pythonhosted.org/cymruwhois/api.html>. [Accessed: 21-Apr-2020].
- [8] A. Tran, P. Lam, and A. Bhangoo, “Ransomware Detection Mechanism,” *Github*. [Online]. Available: <https://github.com/TranAlan/Ransomware-Detection-Mechanism>. [Accessed: 21-Apr-2020].
- [9] “Malware Capture Facility Project,” *Stratosphere IPS*. [Online]. Available: <https://www.stratosphereips.org/datasets-malware>. [Accessed: 21-Apr-2020].
- [10] “Actionable threat intelligence for digital forensics readiness,” *Information and Computer Security*.
- [11] *Protocol Numbers*. [Online]. Available: <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>. [Accessed: 21-Apr-2020].
- [12] K. P. Shung, “Accuracy, Precision, Recall or F1?,” *Medium*, 10-Apr-2020. [Online].

Available: <https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9>. [Accessed: 21-Apr-2020].

- [13] A. Mishra, “Metrics to Evaluate your Machine Learning Algorithm,” *Medium*, 01-Nov-2018.[Online]. Available: <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234>. [Accessed: 21-Apr-2020].
- [14] “sklearn.metrics.average\_precision\_score¶,” *scikit*. [Online]. Available: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average\\_precision\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html). [Accessed: 21-Apr-2020].