Malware Analysis & Classification
Midterm Progress

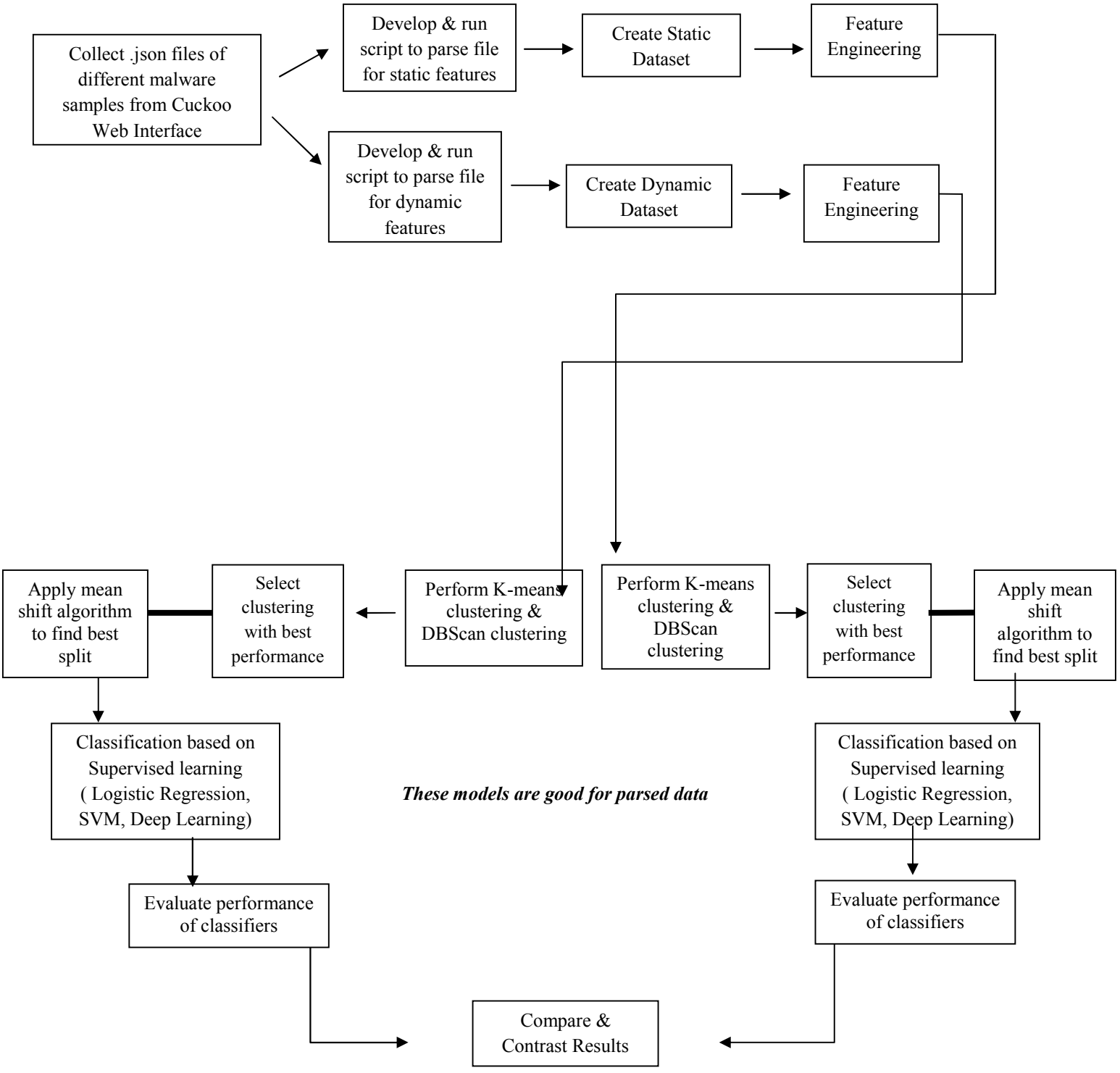
Advisors:

Dr. Matthew Elder, Johns Hopkins University Applied Physics Laboratory
William J. La Cholter, Johns Hopkins University Applied Physics Laboratory
Dr. Xiangyang Li, Johns Hopkins University

Team:

Karol Pierre, Johns Hopkins University
Yu Qiu, Johns Hopkins University
Cheng Xu, Johns Hopkins University
Alex Yang, Johns Hopkins University

Design & Analysis for Parallel Approach **(Hybrid)**



Hybrid Approach

1. Collect .json files of malware samples (200 samples) from Cuckoo,
    - https://cuckoo.cert.ee/analysis/
    - https://sandbox.pikker.ee/
    - Extract static data & dynamic data from these samples

2. Once dataset is collected and prepared, perform clustering using K-means & DBscan (See Appendix A & B)
    - Using two cluster algorithms provides options to choose the better performer
3. Select clustering alg. with best performance to apply mean shift to find the best split
4. Use Logistic Regression, SVM, and Deep Learning for supervised learning for classification
5. Evaluate performance of classifiers
6. Use results from all three algorithms for both dynamic & static and compare and contrast

Evaluation & Result Analysis

In progress . . . The objective is to compare results of Dynamic Analysis Approach with results from Static Analysis Approach and use those comparisons to showcase accuracy or the lack therof.

Current State:

- Completed first round of data collection, extraction, for 50 malware samples for static dataset *(automated process)*
    - Features: Focusing on PE Header Data: Entropy, Virtual Size, Data Size
- Completed first round of data collection, extraction, and cleaning for 50 malwares samples for dynamic dataset --> *needs to be cleaned*
    - Features: API Calls, processes, frequency of api calls
        - Considering using network traffic

Future State

- Collect at least 200 more samples (& Repeat steps listed above)
- Clustering & performance evaluation for both dynamic and static dataset
- Training algorithms (Logistic Regression, SVM, & Deep Learning)
- Result Analysis

Appendices

Appendix A. *JSON to CSV Data Parser Script*

```
import sys
import json
import csv


##
# Convert to string keeping encoding in mind...
##
def to_string(s):
    try:
        return str(s)
    except:
        # Change the encoding type if needed
        return s.encode('utf-8')


def reduce_item(key, value, reduced_item):
    # Reduction Condition 1
    if type(value) is list:
        i = 0
        for sub_item in value:
            reduce_item(key + '#' + to_string(i), sub_item, reduced_item)
            i = i + 1
        return reduced_item

    # Reduction Condition 2
    elif type(value) is dict:
        sub_keys = value.keys()
        for sub_key in sub_keys:
            reduce_item(key + '#' + to_string(sub_key), value[sub_key], reduced_item)
        return reduced item
```

Appendix A. *JSON to CSV Data Parser Script (cont)*

```
    # Base Condition
    else:
        reduced_item[to_string(key)] = to_string(value)
        return reduced_item


def parse(node, path_idx, data_to_be_processed, csv_file_paths):
    print("parsing...")
    processed_data = []
    header = []
    if type(data_to_be_processed) is list:
        for item in data_to_be_processed:
            reduced_item = {}
            reduce_item(node, item, reduced_item)
            header += reduced_item.keys()
            processed_data.append(reduced_item)
    else:
        reduced_item = {}
        reduce_item(node, data_to_be_processed, reduced_item)
        header += reduced_item.keys()
        processed_data.append(reduced_item)

    header = list(set(header))
    header.sort()
    with open(csv_file_paths[path_idx], 'a') as f:
        writer = csv.DictWriter(f, header, quoting=csv.QUOTE_ALL)
        writer.writeheader()
        for row in processed_data:
            writer.writerow(row)
    print("Just completed writing csv file with %d columns" % len(header))
if __name__ == "__main__":
    json_file_path = './Hybrid/report_'
    nodes = ["signatures", "static", "behavior"]
    csv_file_paths = [0] * len(nodes)
    for i in range(len(nodes)):
        csv_file_paths[i] = nodes[i] + ".csv"
        f = open(csv_file_paths[i], 'w')
        f.close()


    for count in range(53):
        print("file: " + str(count))
        fp = open(json_file_path + str(count) + '.json', 'r')
        json_value = fp.read()
        raw_data = json.loads(json_value)
        fp.close()

        if 'pe_sections' in raw_data[nodes[1]] and 'apistats' in raw_data[nodes[2]]:
            parse('signatures', 0, raw_data[nodes[0]], csv_file_paths)
            parse('pe_sections', 1, raw_data[nodes[1]]['pe_sections'], csv_file_paths)
            parse('apistats', 2, raw_data[nodes[2]]['apistats'], csv_file_paths)
```

Appendix B. *Clean the raw .csv file and generate training/testing dataset*

```
import csv
import collections

# this class is used to clean the raw .csv file and generate training/testing dataset
class create_dynamic_set:
    feature_names = set()
    names_values = []

    def __init__(self):
        # open file 1
        try:
            file = open('behavior.csv', 'r')
            csv_reader_lines = csv.reader(file)
        except IOError:
            print('Cannot open file')

        data = []
        for row in csv_reader_lines:
            data.append(row)

        for i in range(0, len(data), 2):
            two_rows = collections.defaultdict(int)
            for j in range(len(data[i])):
                self.feature_names.add(data[i][j].split('#')[-1])
                two_rows[data[i][j].split('#')[-1]] = int(data[i+1][j])
            self.names_values.append(two_rows)

    def outputCsv(self):
        csvFile = open("dynamic_analysis.csv", 'w', newline='')
        print(self.feature_names)
        try:
            writer = csv.DictWriter(csvFile, fieldnames=list(self.feature_names), restval=0)
            writer.writeheader()
            for dic in self.names_values:
                writer.writerow(dic)
            print('The clean data file has been saved')
        finally:
            csvFile.close()

if __name__ == '__main__':
    dynamic_set = create_dynamic_set()
    dynamic_set.outputCsv()
```

Malware Analysis & Classification

Annotated Outline

Advisors:

Dr. Matthew Elder, Johns Hopkins University Applied Physics Laboratory
William J. La Cholter, Johns Hopkins University Applied Physics Laboratory
Dr. Xiangyang Li, Johns Hopkins University

Team:

Karol Pierre, Johns Hopkins University
Yu Qiu, Johns Hopkins University
Cheng Xu, Johns Hopkins University
Alex Yang, Johns Hopkins University

I. Abstract

To be determined . . .

II. Introduction

Problem Statement

Malware is constantly evolving in a fast-paced technological society. As a result, it is imperative that analysis and detection of malware follows the same trend. Accurately analyzing malicious software requires assessing similarities of varying malware types such as worms compared to trojans, in order to determine if there are any relationships or trends.

Purpose of Study

Discover if there any underlying relationships or trends between different malware families and/or malware types

Significance of Research

The ability to accurately detect specific malware signatures or features, will aid in the mitigation and possible development of controls to prevent future strains of malicious software from being able to infect systems

III. Literature Review & Problem Definition

Literature Review

A machine learning approach for Linux malware detection

Classification Problem: Malware Categorization
Type of Input: Goodware binaries, malware binaries
Type of Features: System calls
Algorithm Used: Naïve Bayes, Random Forest

A survey on malware detection using data mining techniques

Classification Problem: Benign vs Malware Categorization
Type of Input: Binary Code
Type of Features: Windows API Calls, N-grams of program code, opcodes, interpretable strings
Algorithm Used**:** Clustering

An investigation of a deep learning based malware detection system

Classification Problem: Benign vs Malware Categorization
Type of Input: Malware assembled code extracted using Unix's objdump and benign executables
Type of Features: Opcodes(frequency), Labels(benign or malware) interpretable strings
Algorithm Used: Deep neural network

Automatic analysis of malware behavior using machine learning

Classification Problem: Behavior
Type of Input: Malware binaries
Type of Features: Changing registry keys or modifying system files
Algorithm Used: Scalable Clustering

HDM-Analyser: a hybrid analysis approach based on data mining techniques for malware detection

Problem: Accuracy in detection
Challenges: Drawbacks of focusing on one type of analysis method
Solution: HDM Analyzer (uses learning model for prediction of code decision making points and executions via API calls from PE & dynamic extraction

Hybrid Analysis and Control of Malware

Problem: Identification
Challenges: Analysis Resistance Techniques - code packing, code overwriting, anti-tampering, anti-emulation checks
Solution: Hybrid Analysis (analyze features of malware binaries and malware execution)

The importance of incorporating both techniques is that it provides the full picture especially in cases where binary data is obfuscated. By analyzing the malware sample and assessing that there is a packer/unpacker, as an example, we gain insight into the complexity and functionality of that type of malware

Malware Detection using Machine Learning and Deep Learning

Classification Problem**:** Benign vs Malware Classification, Binary Classification
Type of Input: disassembled benign executables, disassembled malware binaries
Type of Features: Opcode frequencies, API calls
Algorithm Used: Random Forest

On the feasibility of Malware Authorship Attribution

Classification Problem: Benign vs Malware Classification, Binary Classification
Type of Input: disassembled benign executables, disassembled malware binaries
Type of Features: Opcode frequencies, API calls
Algorithm Used: Random Forest

When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries

Classification Problem: Executable Binary Classification
Type of Input: Executable binaries
Type of Features: Features from decompiled executable binaries
Algorithm Used: Random Forest

Problem Definition

Current Expectations:

Main Focus:

- Through the use of dynamic and static analysis of malware samples we wil be able to classify these samples into families via the application of Machine Learning Algorithms

Sub-Focus:

- *Accuracy of Malware Classification:* the ability to best determine which family each malicious software belongs to, in order to assist with best methods for malware mitigation
- Dynamic vs Static Analysis: Comparison of classification results

IV. Technical Solution, Design and Analysis

Technical Solution

Methods Considered

Design & Analysis for Dynamic Approach (1)

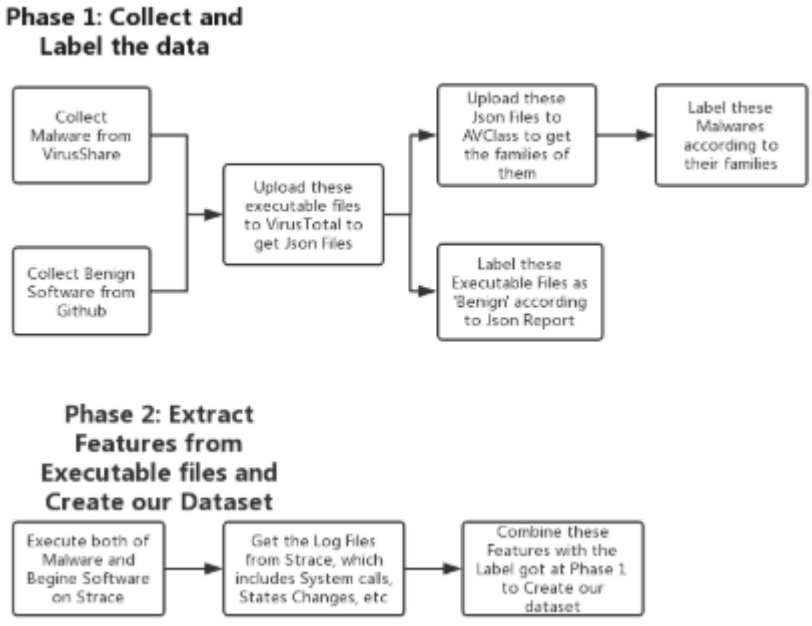


Figure 1: 2-Phase Malware Attribution Approach

The flow chart in Figure 1 showcases the two-pronged approach that was first considered. This approach was unsuccessful due to lack of available executable Linux malware. There was a plethora of Linux malware binaries that could be used to obtain hash values and attempt to label using the tools VirusTotal and AVclass. Collection and

labeling of data in this method proved to be unsuccessful due to the lack of compatibility between VirusTotal and AVclass. As a result, phase two: extraction of features and development of the dataset is no longer feasible. This forces a creation of a new methodology to meet the same objective of collecting and labeling data and feature and extraction and dataset development.

Methods Considered *(cont)*

Design & Analysis for Dynamic Approach (2)

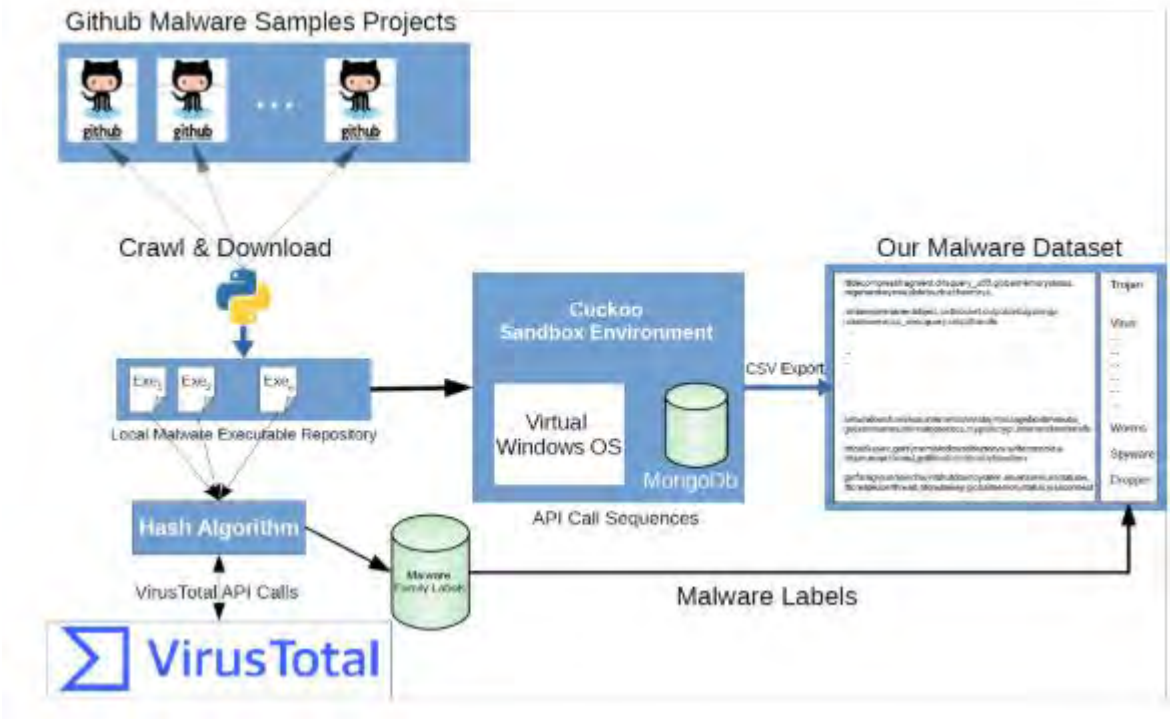


Figure 2: Catak, FÖ. and Yazi, AF. 's API Calls Dataset Acquisition

The flow chart in Figure 2 depicts the dynamic approach that was used by Catak and Yazi. This dynamic approach which includes the execution of malware in an isolated environment, a sandbox (Cuckoo), produced a dataset that consists of API calls extracted from monitoring the process executed while malware was executed. The derived dataset was used to classify different malware into approximately 8 categories: Trojan, Backdoor, Worms, Spyware, Adware, Dropper, Virus.

While this approach is great for pure data analysis, it would not be sufficient for the desired end goal of comparing and contrasting results of Dynamic Analysis using the Catak/Yazi dataset with the results of the Static Analysis using the Te-k/Malware dataset. The dataset needs to contain the same malware samples in order to accurately assess and measure the similarities, differences and level of accuracy of malware classification

Methods Considered *(cont)*

Design & Analysis for Static Approach

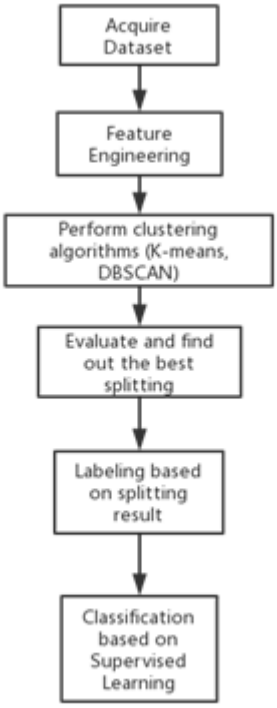


Figure3: Desired Malware Attribution Approach

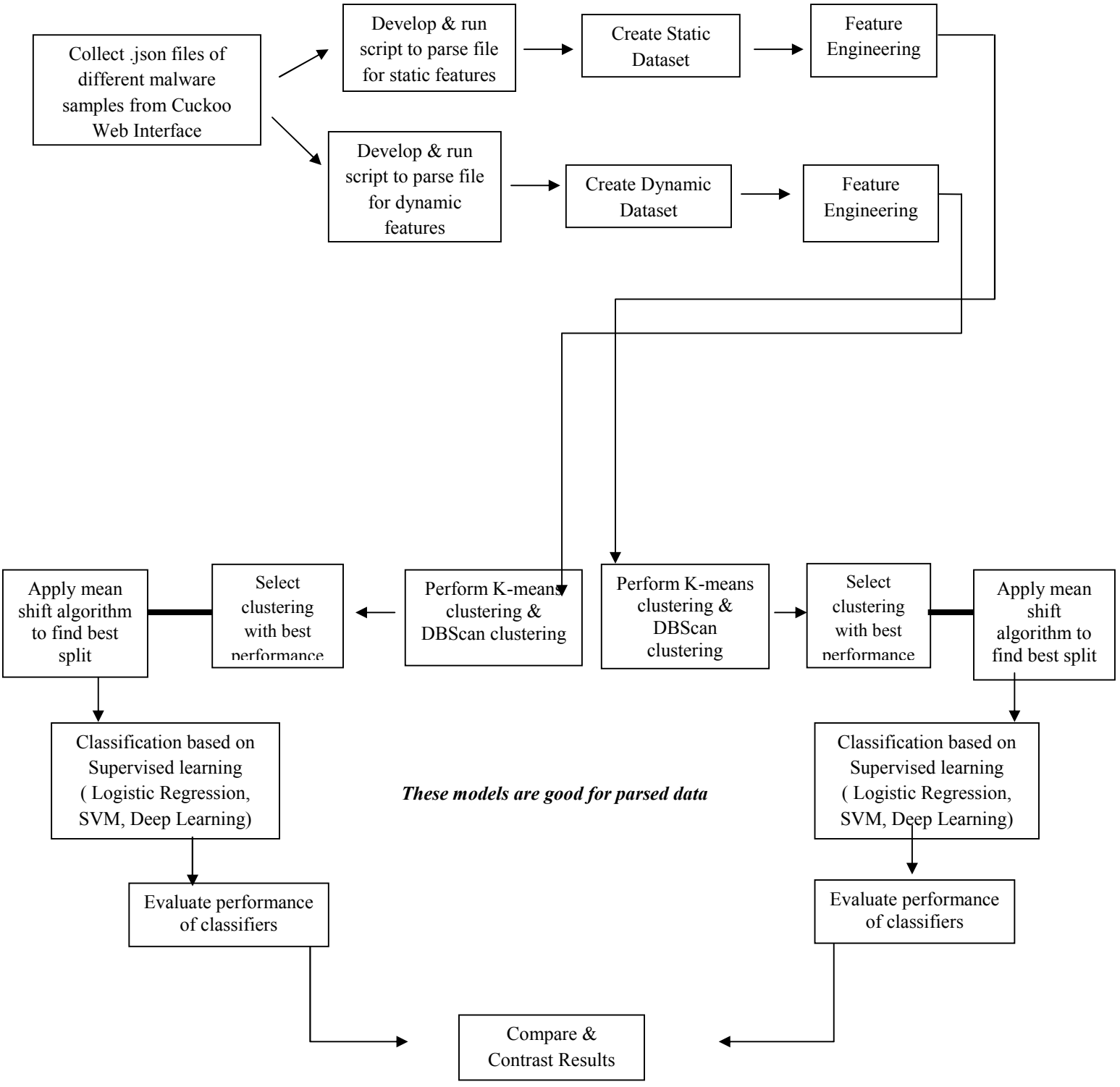
The flowchart in Figure 3 showcases our chosen approach for static analysis. Using the dataset acquired from , https://github.com/Te-k/malware-classification , and by conducting feature engineering on the dataset, the preprocessing of the dataset was completed. By implementing the K-means algorithm for clustering, 5 clusters (*significance to be determined)* were produced:

- In the process of interpreting cluster significance (*Labeling based on splitting result)*
- Final step before supervised learning to develop classification
- Performance of clustering is not guaranteed to be high

While this approach is also great for pure data analysis, it would not be sufficient for the desired end goal of comparing and contrasting results of Dynamic Analysis using the Catak/Yazi dataset with the results of the Static Analysis using the Te-k/Malware dataset. The dataset needs to contain the same malware samples in order to accurately assess and measure the similarities, differences and level of accuracy of malware classification

Methods Selected

Design & Analysis for Parallel Approach (Hybrid)



These models are good for parsed data

V. Experimentation, Evaluation and Result Analysis

Hybrid Approach

1. Collect .json files of malware samples (200 samples) from Cuckoo,
   - https://cuckoo.cert.ee/analysis/
   - https://sandbox.pikker.ee/
   - Extract static data & dynamic data from these samples
2. Once dataset is collected and prepared, perform clustering using K-means & DBscan (See Appendix A & B)
   - Using two cluster algorithms provides options to choose the better performer
3. Select clustering alg. with best performance to apply mean shift to find the best split
4. Use Logistic Regression, SVM, and Deep Learning for supervised learning for classification
5. Evaluate performance of classifiers
6. Use results from all three algorithms for both dynamic & static and compare and contrast

Evaluation & Result Analysis

In progress . . . The objective is to compare results of Dynamic Analysis Approach with results from Static Analysis Approach and use those comparisons to showcase accuracy or the lack therof.

VI. Conclusion

To be determined . . .

VII. References

Alrabaee, Saed, et al. "On the feasibility of malware authorship attribution". *International Symposium on Foundations and Practice of Security*. Springer, Cham, 2016.

Asmitha, K A, and P Vinod. "A Machine Learning Approach for Linux Malware Detection." *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014, doi:10.1109/icicict.2014.6781387.

Caliskan, Aylin, et al. "When coding style survives compilation: De-anonymizing programmers from executable binaries." *arXiv preprint arXiv:1512.08546* (2015).

Eskandari, Mojtaba, Zeinab Khorshidpour, and Sattar Hashemi. "HDM-Analyser: a hybrid analysis approach based on data mining techniques for malware detection." *Journal of Computer Virology and Hacking Techniques* 9.2 (2013): 77-93.

Microsoft. "PE Format."*WIndows Dev Center*. https://docs.microsoft.com/en-us/windows/win32/debug/pe-format#file-headers

Rathore, Hemant, et al. "Malware Detection Using Machine Learning and Deep Learning." *International Conference on Big Data Analytics*. Springer, Cham, 2018.

Rieck, Konrad, et al. "Automatic Analysis of Malware Behavior Using Machine Learning." *Journal of Computer Security*, vol. 19, no. 4, 2011, pp. 639–668., doi:10.3233/jcs-2010-0410.

Roundy, Kevin A., and Barton P. Miller. "Hybrid analysis and control of malware." *International Workshop on Recent Advances in Intrusion Detection*. Springer, Berlin, Heidelberg, 2010.

Sewak, Mohit, et al. "An Investigation of a Deep Learning Based Malware Detection System." *Proceedings of the 13th International Conference on Availability, Reliability and Security - ARES 2018*, 2018, doi:10.1145/3230833.3230835.

Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. ACM Comput. Surv. 50, 3, Article 41 (June 2017), 40 pages. DOI: http://dx.doi.org/10.1145/3073559

Yazı, Ahmet & Catak, Ferhat Ozgur & Gul, Ensar. (2019). Classification of Methamorphic Malware with Deep Learning (LSTM). 10.1109/SIU.2019.8806571.

VIII. **Appendices**

Appendix A. *JSON to CSV Data Parser Script*

```
import sys
import json
import csv


##
# Convert to string keeping encoding in mind...
##
def to_string(s):
    try:
        return str(s)
    except:
        # Change the encoding type if needed
        return s.encode('utf-8')


def reduce_item(key, value, reduced_item):
    # Reduction Condition 1
    if type(value) is list:
        i = 0
        for sub_item in value:
            reduce_item(key + '#' + to_string(i), sub_item, reduced_item)
            i = i + 1
        return reduced_item

    # Reduction Condition 2
    elif type(value) is dict:
        sub_keys = value.keys()
        for sub_key in sub_keys:
            reduce_item(key + '#' + to_string(sub_key), value[sub_key], reduced_item)
        return reduced item
```

Appendix A. *JSON to CSV Data Parser Script (cont)*

```
    # Base Condition
    else:
        reduced_item[to_string(key)] = to_string(value)
        return reduced_item


def parse(node, path_idx, data_to_be_processed, csv_file_paths):
    print("parsing...")
    processed_data = []
    header = []
    if type(data_to_be_processed) is list:
        for item in data_to_be_processed:
            reduced_item = {}
            reduce_item(node, item, reduced_item)
            header += reduced_item.keys()
            processed_data.append(reduced_item)
    else:
        reduced_item = {}
        reduce_item(node, data_to_be_processed, reduced_item)
        header += reduced_item.keys()
        processed_data.append(reduced_item)

    header = list(set(header))
    header.sort()
    with open(csv_file_paths[path_idx], 'a') as f:
        writer = csv.DictWriter(f, header, quoting=csv.QUOTE_ALL)
        writer.writeheader()
        for row in processed_data:
            writer.writerow(row)
    print("Just completed writing csv file with %d columns" % len(header))
if __name__ == "__main__":
    json_file_path = './Hybrid/report_'
    nodes = ["signatures", "static", "behavior"]
    csv_file_paths = [0] * len(nodes)
    for i in range(len(nodes)):
        csv_file_paths[i] = nodes[i] + ".csv"
        f = open(csv_file_paths[i], 'w')
        f.close()


    for count in range(53):
        print("file: " + str(count))
        fp = open(json_file_path + str(count) + '.json', 'r')
        json_value = fp.read()
        raw_data = json.loads(json_value)
        fp.close()

        if 'pe_sections' in raw_data[nodes[1]] and 'apistats' in raw_data[nodes[2]]:
            parse('signatures', 0, raw_data[nodes[0]], csv_file_paths)
            parse('pe_sections', 1, raw_data[nodes[1]]['pe_sections'], csv_file_paths)
            parse('apistats', 2, raw_data[nodes[2]]['apistats'], csv_file_paths)
```

Appendix B. *Clean the raw .csv file and generate training/testing dataset*

```
import csv
import collections

# this class is used to clean the raw .csv file and generate training/testing dataset
class create_dynamic_set:
    feature_names = set()
    names_values = []

    def __init__(self):
        # open file 1
        try:
            file = open('behavior.csv', 'r')
            csv_reader_lines = csv.reader(file)
        except IOError:
            print('Cannot open file')

        data = []
        for row in csv_reader_lines:
            data.append(row)

        for i in range(0, len(data), 2):
            two_rows = collections.defaultdict(int)
            for j in range(len(data[i])):
                self.feature_names.add(data[i][j].split('#')[-1])
                two_rows[data[i][j].split('#')[-1]] = int(data[i+1][j])
            self.names_values.append(two_rows)

    def outputCsv(self):
        csvFile = open("dynamic_analysis.csv", 'w', newline='')
        print(self.feature_names)
        try:
            writer = csv.DictWriter(csvFile, fieldnames=list(self.feature_names), restval=0)
            writer.writeheader()
            for dic in self.names_values:
                writer.writerow(dic)
            print('The clean data file has been saved')
        finally:
            csvFile.close()

if __name__ == '__main__':
    dynamic_set = create_dynamic_set()
    dynamic_set.outputCsv()
```