NORTHWEST NAZARENE UNIVERSITY

Identifying ATT&CK® Tactics in Android Malware Control Flow Graph Through Graph Representation Learning and Interpretability.

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Jeffrey G. Fairbanks
2021

THESIS
Submitted to the Department of Mathematics and Computer Science
in partial fulfillment of the requirements
for the degree of
BACHELOR OF SCIENCE

Jeffrey G. Fairbanks
2021

Identifying ATT&CK® Tactics in Android Malware Control Flow Graph Through Graph Representation Learning and Interpretability.

Author: Jeffrey Fairbanks
Jeffrey G. Fairbanks

Approved: Dale A Hamilton
Dr. Dale Hamilton. Ph.D., Department of Mathematics and Computer Science, Faculty Advisor

Approved: Donna M. Allen
Dr. Donna M. Allen, Department of Communication Arts & Science, Second Reader

Approved: Barry Myers
Barry L. Myers, Ph.D., Chair, Department of Mathematics & Computer Science

Abstract

Identifying ATT&CK® Tactics in Android Malware Control Flow Graph Through Graph Representation Learning and Interpretability.

FAIRBANKS, JEFFREY (Northwest Nazarene University Department of Mathematics & Computer Science)

Malware affects millions of machines, causing havoc to those it reaches. The dangers and negative impact that malware has inflicted push researchers to find a way to mitigate its effects. Labeling malware within the anti-malware services becomes a challenge in finding the correct Tactics, Techniques, and Procedures (TTP) that each malware implements. The Control Flow Graph (CFG) describes the structure of a program during its execution; this is how a program flows. In reference to malware, it represents the flow of all the internal and external function calls. The current research proposes a novel approach to locating ATT&CK® TTP in a CFG by applying Machine Learning Classifiers on Android Malware. Through these methods, the approach associates the TTP, given by the ATT&CK® Framework with a subgraph of an Android malware CFG. Using Graph Neural Network and SIR-GN node representation learning approach, this methodology processes the CFG and creates a model that classifies the associated TTP. Furthermore, the explanation technique SHapley Additive exPlanations (SHAP), a model agnostic game-theoretic approach to explain any machine learning model's output and identify the subgraph in the CFG connected with the specific TTP, is implemented. Preliminary experiments indicate approximately 89% accuracy in classifying such techniques.

Keywords

Machine Learning, Tactics Techniques, and Procedures, Control Flow Graph, Android Malware, Graph Isomorphic Network, Random Forest Classifier, SHapley Additive exPlanations, Malware Analysis, MITRE ATT&CK®, Hybrid-Analysis Sandbox, SIR-GN

Acknowledgments

This research was made possible by the National Science Foundation for funding this research (CCF 1950599) and Boise State University for hosting the Research Experience for Undergraduates (REU) program.

This research was completed with the assistance of Andres Orbe (New Jersey Institute of Technology), Christine Patterson, Dr. Edoardo Serra, and Dr. Marion Scheepers (Boise State University). This thesis will focus on those elements to which I was the main contributor but note that this work was done with the assistance of the contributors mentioned above.

Table Of Contents

Table Of Figures

1 Introduction

Malware results in substantial monetary damages, leading to loss of trust in a business and possibly crippling entire organizations. To mitigate these threats, one must have a deep understanding of malware and face the challenge of going against bad actors. With every new way to detect malware, there is equally a new way to avoid detection. This creates tensions between those seeking to mitigate threats and bad actors who wish to cause harm by promulgating malware. To facilitate a safer environment for those connected to the internet, this research turns to machine learning to help automatically detect techniques being used within Android Malware. Currently, no automated procedures can characterize, given the malware executable, what part of the execution flow relates to specific Tactics, Techniques, and Procedures (TTP) that the malware utilizes. This research provides an automation methodology to locate specific TTP in a sub-part of the Control Flow Graph that describes the execution flow of a malware executable. Finding the TTP that the malware implements allows for better mitigation, as the attack vectors of the malware can be clearly seen. This process is achieved through graphing malware by extracting its Control Flow Graph (CFG) and mapping it to the specific Tactics, Techniques, and Procedures the malware is using that are described through the MITRE ATT&CK® Framework [14].

In Figure 1 below, the right subgraph, denoted by the red box enclosing it, correlates directly to a Tactic 'Impact' and the Technique 'Carrier Billing Fraud'. Also in the figure, the dashed-blue line correlates to the Tactic 'Collection' and the Technique 'Access Contact List'. These Tactics Techniques and Procedures (TTP) seen on the right come from the MITRE ATT&CK® Framework.

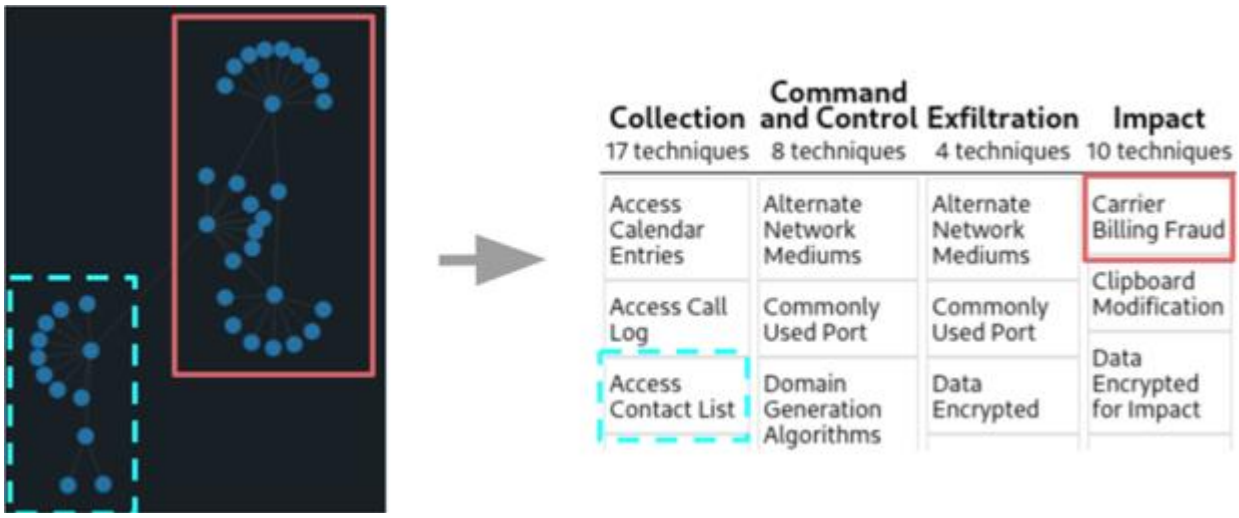

Figure 1

Each Subgraph maps to the specified TTP as determined by the color of the box surrounding it.

The association of a specific TTP to the subgraph that it is contained in is important to better footprint malware and create better safeguards against malware in the future.

2 Background

Several research papers have used a variety of methods to study the detection of malware by using graphs. Mehadi Hassen et al. have contributed to research in the area of malware classification using Neural Networks with their paper, *Scalable Function Call Graph-based Malware Classification* [13]. By using Machine Learning and a Control Flow Graph, they were able to correctly identify whether a given Android Malware was malicious. However, this research does not include the classification of which techniques are being used in the given malware or where these techniques are located within the Control Flow Graph. Similar work was also completed by Xin Hu et al. in their research, *Large-Scale Malware Indexing Using Function-Call Graphs* [27], as well as the work by Mojtaba et al. in their paper, *Metamorphic Malware Detection using Control Flow Graph Mining* [12]. While each of these research efforts has made strides in malware detection, the classification of malware through machine learning can be widely improved upon.

MITRE ATT&CK® (Adversarial Tactics, Techniques, and Common Knowledge) [14] has labeled a number of popular malware families with their given TTP; however, it does not explain the location where the TTP exists in the Malware's CFG. MITRE has been essential in the development of this research as this organization has created a database of TTP, that is used to aid in classifying malware. With this given TTP, each malware can be given some subset of these TTP based on the CFG and the behavior of the malware, allowing for the labeling of the data.

2.1 MITRE ATT&CK®

MITRE ATT&CK® [14] is a globally accessible framework of adversary tactics and techniques based on real-world observations. ATT&CK® was developed in response to the challenge of defending against Advanced Persistent Threats (APT) and has led to several valuable research on the detection and classification of malware [2]. Through ATT&CK®, TTP is more easily detected because the TTP is readily available in the repository. The ATT&CK® framework is used as a foundation for the development of specific threat models and methodologies in the private sector, the government, and the cybersecurity product and service community.

The development of MITRE ATT&CK® has resulted in further research into TTP and malware. A. Georgiadou et al. have written a research paper, *Assessing MITRE ATT&CK® Risk Using a Cyber-Security Culture Framework*, where they use ATT&CK® to explore the organization of security procedures and the improvement of security risk comprehension [1]. This was accomplished through the analysis of associating a comprehensive set of organizational and individual culture factors while utilizing the MITRE ATT&CK® framework to map security vulnerabilities with specific adversary behaviors and patterns [1]. The MITRE ATT&CK® framework has also been utilized in the work described in *Automated Threat Report Classification Over Multi-Source Data* written by A. Gbadebo et al [6]. This research focuses on the extraction of adversary actions from threat report documents and automatically classifies them into tactics and techniques.

MITRE ATT&CK® is an ever adapting and expanding knowledge base for designing threat models. In the knowledge base, there are many different functionalities that benefit this research. First, ATT&CK® [14] highlights several popular malware classes, or *Software*, that contain a list of Tactics, Techniques and Procedures (TTP). It is important to note that malware tends to fall into groups of this software, called *Malware Families*. New malware makes slight modifications from the original version of itself in an attempt to thwart programs designed to protect against malware. ATT&CK® not only keeps track of groups of software but also tracks groups of malware developers. Furthermore, ATT&CK® labels each of these Malware Families with a set of TTP that are commonly seen. Given these TTP, researchers can understand the attack vectors employed.

ATT&CK® has both enterprise and mobile-level TTP. In this research, the Mobile TTP is used. This mobile TTP contains information for Android and IOS devices. For the purposes of this research, the Android TTP is used. The TTP listed under the Mobile Section are the malicious actions that a malware undertakes to gain access to an Android device. These tactics include Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration, and Impact. These are not all implemented within the samples that are collected through this research, and some are left out throughout the completion of the research project [15].

2.2 Control Flow Graph

A Control Flow Graph (CFG) gives the graphical representation of the structure of a program. This works by graphing the functions that the program calls as it executes. Through this method, one can better visualize what is happening inside the inner workings of the program, as well as determine how each function connects to one another. These functions can be external functions, such as API calls, or can be internal functions included in the packages that the developer adds into the workspace, which can also be the internal functions written by the programmer themselves. Furthermore, CFGs are helpful to show the Entry Block and Exit block of the code, allowing one to see where the code begins and ends; this aids in the reverse engineering of malware.

The use of CFGs in research involving malware is common, as CFGs are valuable to the visualization and understanding of how a program works. Guillaume Bonfante et al. have contributed to the work surrounding the use of CFGs in the detection of malware in their paper, *Control Flow Graphs Malware Signatures* in which they extract CFGs from malware and use it as signatures for malware detection. Another paper written by A. Kapoor et al., *Control Flow Graph Based Multiclass Malware Detection Using Bi-normal Separation*, uses bi-normal separation to detect malware [3]. This research was proposed to aid in the classification of malware into their malware families, as much of the research completed at that point focused purely on the detection of malware [3]. Another publication regarding CFGs in malware detection written by S. S. Anju et al, *Malware Detection using Assembly Code and Control Flow Graph Optimization*, presents a method of malware detection that is resilient to common obfuscation transformations [22]. Malware detection in their research is conducted by finding signatures of the malware to get the syntactic characteristics; however, small changes can be made to malware that makes this detection

method less effective. In their research, they focus on using syntactic and semantic features to detect malware by using CFGs, thus making it a more reliable malware detection method [22].

Figure 2

A simple Control Flow Graph showing external API calls and internal calls

Figure 2 shows an example of a Control Flow Graph (CFG). Each CFG utilized in this research is a directed graph visualizing the flow of a program. The internal functions, whether they are written by the programmer or packages imported by the programmer, are denoted as a gray box with the name 'sub' with numbers following. The external functions, such as the Android API calls, are denoted as a light blue box. These API calls are calls that are made to the Android API library.

One example of an API call is the call named "getDeclaredField()". This external API call returns a 'field' object for use within the flow of the program.

3 Tool Selection

After understanding the constraints and planning the outcomes of the research, the researcher explored tools to use in this novel approach to Graph-Based Malware Analysis. As the research progressed specific tools were implemented but then switched out with others that provided increased performance. Tools were selected based on the given criteria:

- Cost
- Documentation
- API usability/availability
- Ability to analyze malware dynamically or statically

In the early stages, while conducting preliminary research, a few tools were found to be beneficial and were selected. The chosen pipeline was also selected because of the intent to conduct research on Windows Portable Executable (PE) malware. The first pipeline used early in the project can be seen in the Figure 3.

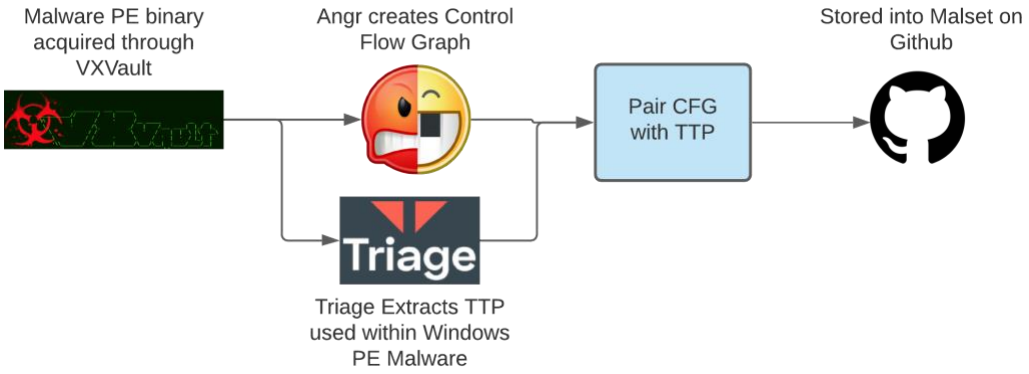

Figure 3

The preliminary pipeline used to extract CFG and ATT&CK® TTP

In this pipeline, there are two main tools utilized to prepare the malware for classification. One tool allows for the parsing of the CFG, and the other tool performs analysis of the malware in a sandbox to provide ATT&CK® Tactics, Techniques and Procedures (TTP) that are implemented within the malware. The first tool, Angr [29], is a platform-agnostic binary analysis framework that conducts static analysis on Windows Portable Executable (PE) files. Hatching Triage Sandbox [31] is then used to analyze the binary executable sample and extract the ATT&CK® TTP that the malware implements.

The utilization of conducting research on Windows PE malware quickly changed, as it was found that conducting research using static analysis tools such as Angr is less effective in rapidly evolutionary and dynamic threat space and can be argued as controversial within the scientific community [32]. Following this discovery, the project was altered to conduct research on Android malware in the form of .apk instead of Windows PE files. As this move was made, the tools previously used no longer conducted the proper analysis, and new tools had to be explored. For example, Angr [29], conducts exemplary static binary analysis on windows PE files, allowing for the formation of the CFG, but does not perform adequately on Android Malware. Angr is unable to create a CFG of an Android .apk file. In the same way, Hatching Triage Sandbox [31] does not return any information of the Tactics, Techniques and Procedures (TTP) being implemented for any given Android Malware. When a malware is sent into the sandbox, the hash is analyzed and the TTP that were previously assigned manually are returned. However, when using Hatching Triage Sandbox with Android Malware no TTP are assigned, as Hatching Triage Sandbox focuses solely on Windows malware.

In the search for new tools that could handle analysis on Android executable (.apk) files, two new tools were found to be effective. Androguard [25] was adapted to begin the process of extracting Control Flow Graphs (CFG) from the Android .apk malware. Androguard runs best when on a Linux OS, thus a Linux box was set up for the purpose of extracting the CFG from each

of the thousands of Android Malware .apk. The search for a new Sandbox to retrieve ATT&CK® TTP that the Android Malware was implementing took some time, but ultimately Hybrid Analysis Sandbox [5] was utilized. Hybrid Analysis Sandbox is an Automated Malware Analysis and Sandbox tool powered by CrowdStrike's Falcon Sandbox. With this sandbox comes accessibility to a researcher's API. Hybrid Analysis can inspect the Android Malware executable (.apk file) and return information on which Tactics, Techniques and Procedures (TTP) are being implemented.

The collection of malwares also had to be changed, as the number of samples acquired daily was insufficient and gave no real way of being sure that a sample was only acquired once and in the correct Android .apk form. In the preliminary investigation, while utilizing Windows PE malware, the collection process was done by scraping vxvault.net [30]. With the move to Android .apk malware and the need for better collection standards, VirusTotal [24] was utilized as a database for the collection of Android .apk files. Through VirusTotal, Android .apk files could be acquired and sorted by their SHA256 hashes, allowing for improved collection and database management. The following pipeline, seen in Figure 4, was then constructed, and used for the remainder of the project.

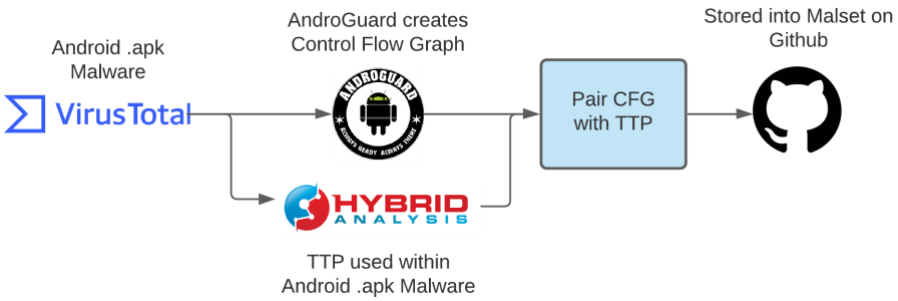

Figure 4

The final pipeline used to extract CFG and ATT&CK® TTP

4 Overview

In this research, a novel approach is explored to Classify TTP within a Control Flow Graph and explain in which subgraph that TTP is contained. There are many processes that must be implemented to fulfill this research. The project can be conceptually divided into three implementation phases: Malware Data Collection, Malware Analysis, and Machine Learning / Explanation.

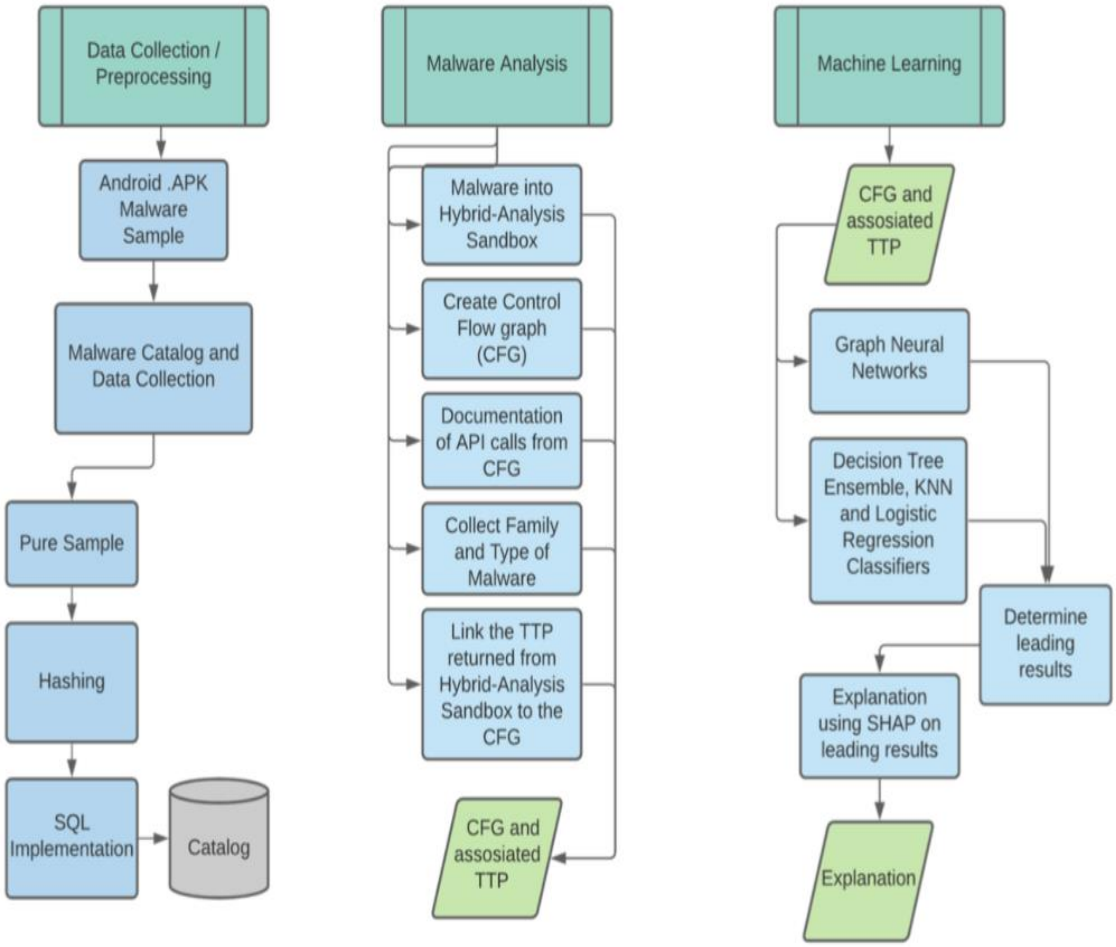

Figure 5

Overview of each part of the implementation

The data collection phase is important to the completion of the rest of the project. Android executable malware was collected and cataloged into a database. After collection, a SHA256 hash was made, along with others, and stored as the name of the malware for easy handling of each specific malware. This data was then used in the next phase of the project.

In the Malware Analysis Phase, the data was taken and given meaning in the context of this research. A Control Flow Graph (CFG) was made from the Android executable using the tool Androguard [25], and the executable was put through Hybrid-Analysis Sandbox [5] to extract the MITRE ATT&CK® Tactics, Techniques and Procedures (TTP) utilized within the malware executable. Furthermore, the type of malware and family associations were extracted for later use after classification and explanation had been completed. Finally, the TTP and the CFG are linked together and saved under the name of the specific Android Malware hash. This file was then sent to pre-processing to be given to the Machine Learning / Explanation Phase.

In the final phase of the research, the CFG and linked TTP are now cleaned and pre-processed, ready to be passed through the Machine Learning Classifiers. After classification, the leading results are chosen and passed onto the explanation. This part of the research explains in which subgraph of the entire CFG that a specific predicted TTP is contained.

5 Data Collection and Analysis

Data collection consists of binary, or executable, Android Malware in the form of an .apk file. The format of these files must be Android .apk to pull all the necessary information. There are complications to finding Android samples, as executable Malware samples are typically not available to the public. VirusTotal [24] is an application that allows for the acquisition and analysis of Malware samples. Through VirusTotal, the executable Android Malware is acquired, increasing the data set up to more than 8000 samples. These raw samples must then be in the proper format to be passed into the machine learning process.

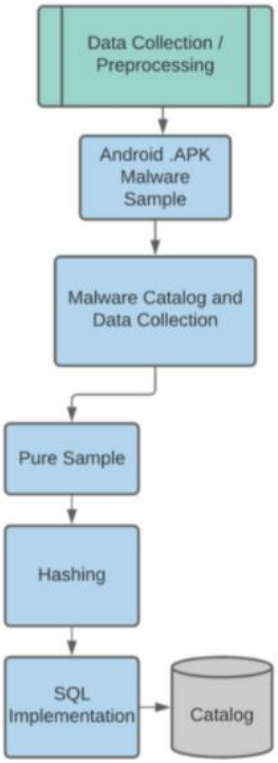

Figure 6

Overview of the first phase of implementation

After the collection and cleaning of the data from VirusTotal, data is passed into Hybrid-Analysis Sandbox [5] to collect the ATT&CK® TTP. Many options for the sandbox did not include the information necessary to extract the TTP from the data. Hybrid-Analysis Sandbox [5], also known as Falcon Sandbox, allows the raw Android .apk executable samples to be passed in, and extracts the ATT&CK® TTP that the malware executable contains. The data collection proceeds in this manner for some time as the Researchers API only allows for 100 uploads to the sandbox every 24 hours. Through the implementation of Hybrid-Analysis Sandbox, the TTP can be gathered.

Lastly, the CFG is extracted from the collected samples by using AndroGuard [25] that extracts and creates the CFG from the Android .apk file.

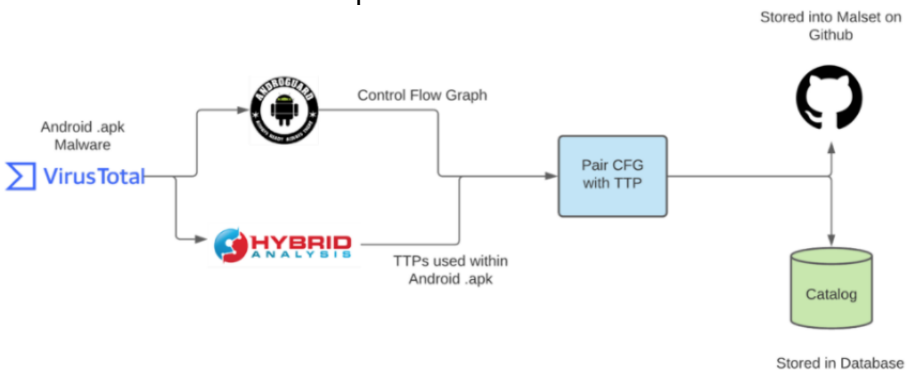

Figure 7

The executables go through Hybrid Analysis Sandbox to extract the TTP.

AndroGuard is used to extract the CFG. The TTP and CFG are paired and cataloged.

During the process of collecting data, a catalog is made to ensure that each sample is processed exactly once with no duplicates. This process is also helpful in ensuring the data consists of a graph and contains relevant information regarding the TTP before continuing to the machine learning process.

5.1 Documenting Android API calls

Where each node in the Control Flow Graph (CFG) is a function call, there must be a way to understand what each function call is completing, and specifically external API calls. This led to the development of a web crawler to grab the documentation for each Android API call. A python script was developed to crawl the Android API Documentation [4] [16]. By completing the task of gathering information and making it easily available during the explanation process, each API call and its purpose are cataloged.

For example, in one node of the Control Flow Graph (CFG), an external API function call was made. This call was an android API call 'getDeclaredField()'. (see Figure 8)

```
node [
  id 6
  label "Ljava/lang/Class;->getDeclaredField(Ljava/lang/String;)Ljava/lang/reflect/Field;"
  external 1
  entrypoint 0
]
```

Figure 8

External API call "getDeclaredField" is called in this node of one CFG

Using the Android API web crawler, this API call can be documented and explained to show the purpose of a specific API call. In this case, the API call made was to retrieve a 'field' object for the use of the malware. (see Figure 9)

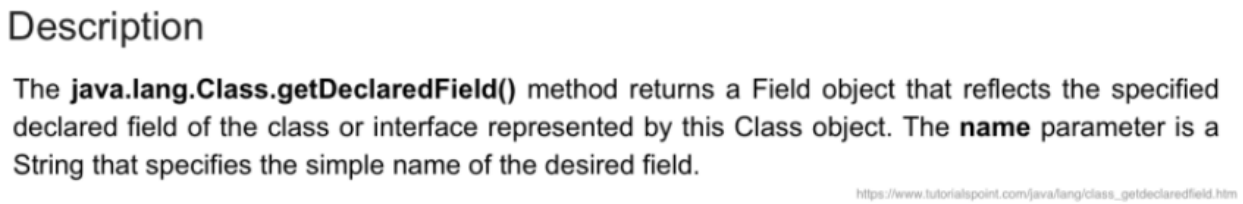

Figure 9

Explanation of the API call "getDeclaredField"

This work is important in the explanation of each TTP within a Control Flow Graph as it helps validate the results. As each API call is documented, the problem does not lie in one single API call. It is only when these calls are made together that they form a subgraph within the CFG as the malware implements a certain TTP. For example, Finding an API call getDeclaredField() is not a problem in itself, but this coupled with other API calls can create a subgraph where the specific TTP “Discovery“ is being used.

6 Machine Learning Experimentation

After the data is collected, cleaned, processed, and cataloged, a Graph Isomorphic Neural Network (GIN), and a Graph Attention Neural Network (GAT), built from tools made available through PyTorch's Deep Graph Library [18], are used to begin the machine learning process. In addition to the Neural networks, Decision Tree Ensembles are also used to improve the results. These include a Decision Tree, Random Forest, and ExtraTrees Classifier. Using the ATT&CK® TTP as labels, and the CFG as the input data, the Classifiers will return an F1 Score, which is the harmonic mean of the precision and recall, and the Accuracy Score, which is the number of correct labels predicted divided by the total number of labels used, of each model on a test set. After the completion of training and testing of the Classifiers, the explanation begins using the tool SHAP [11]. The implementation of SHAP gives insight into the model and returns a result showing which TTP is most important for each edge of the CFG and is further adapted to show in which subgraphs of the entire CFG where each TTP is predicted to be located. An overview of this pipeline is seen in Figure 10:

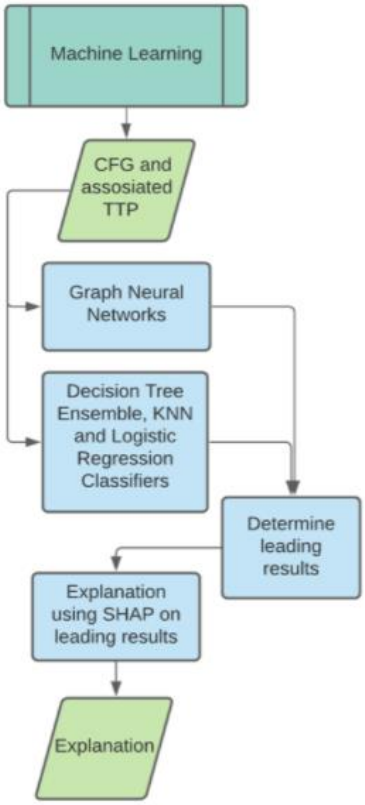

Figure 10

Machine Learning phase 3 overview

Through experimentation of the machine learning results, several observations have come to light regarding classification techniques, quantitative explanation, and qualitative explanation.

6.1 Graph Neural Network Classification

Graph Neural Networks (GNN) are one of the Machine Learning Classifiers used in this project for graph classification. With the data having both a Control Flow Graph (CFG) and Tactics, Techniques, and Procedures (TTP) that the Android Malware executable implements, this data is used as inputs into the GNN. Upon implementing Neural Networks, poor predictive performance was observed. Both Graph Attention (GAT) and Graph Isomorphic (GIN) Neural Network architectures were implemented. These are the best choices for this research specifically because of the graph-based classification implemented. GAT keeps graph structure through its classification strategy and GIN keeps node order intact, both of which are important for CFG classification. Each of these Neural Networks implemented hidden layers in their classification technique. The GIN implemented its hidden layers coupled with Multi-Layer Perceptron (MLP) layers, while the GAT was coupled with the hyperparameter “Number of Heads”.

6.2 Decision Tree Ensemble Classification Using SIR-GN

A group of individual decision trees, called a Random Forest, can be used to classify and demonstrate the explainability of the machine learning model. Each tree predicts an outcome of the class, and the prediction that occurs most often among the Random Forest is the model's class prediction. With this Random Forest Classifier, the TTP is predicted to be in use or not for each CFG. The Random Forest Classifier used in this project is an adaption of the Scikit Learn Library's ExtraTrees Classifier. This Classifier implements a meta estimator that fits several randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting [21]. To implement this classification technique using Scikit Learn ExtraTrees Classifier, the data had to be refitted through the implementation of SIR-GN node representation learning approach [9]. The vectorial representation of SIR-GN provides a procedure to create a unique graph representation technique. Such techniques identify groups of nodes in a fixed number and each group contains nodes with similar vectorial representations. Given this set of groups the method creates a pseudo adjacency matrix working on the groups that, once flattened, represent the vectorial representation of the graph. Then, the vectorial representations of two graphs are comparable if the computation of the node representations and the definition of the groups of nodes or the pseudo adjacency matrices are identical for the two graphs [9]. To guarantee this property, this research uses inferential SIR-GN which is a procedure able to perform inferences and that is pretrained on a specific family of directed random graphs. Note that since the groups are created based on structural similarities among the nodes, the graph representation is invariant under permutation of the nodes in the graph. This methodology assures a fast and comparable creation of graph vectorial representation. Once the vectorial representation of each graph is created, a standard machine learning model can be utilized to classify the presence of specific TTP. The main technique used is the ExtraTrees classification algorithm as this algorithm gives the best classification performance. It is important to note that Graph Neural Networks can achieve the same task. However, it is experimentally demonstrated that they do not perform as well as SIR-GN.

7 Classification Results

The results given by the Classifiers can be seen in this section. Through the process of training and testing on multiple models, this research demonstrates the effectiveness of the novel approach of explaining the Tactics, Techniques, and Procedures (TTP) in a Control Flow Graph (CFG). The first approaches are done using Graph Neural Networks for classification. Through the completion of this research, there are a few different architectures that are used to find the maximum F1 Score and Accuracy. A Graph Isomorphic Neural Network (GIN) and Graph Attention Network (GAT) are implemented using an adaption of the Deep Graph Library implementation (DGL) [26]. DGL allows for the specific tuning and adjusting of specific parameters to allow for effective use on different datasets. Following these methods, a Decision Tree Ensemble and Regression approach are also taken to find an improved solution.

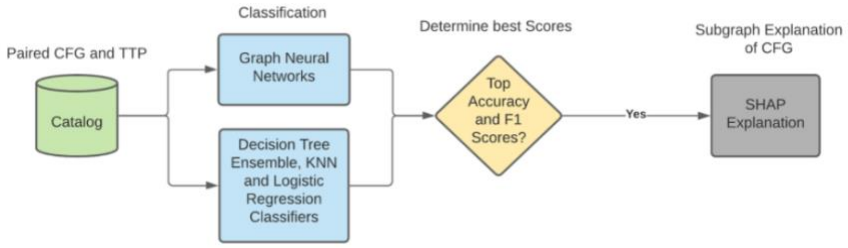

Figure 11

After inputting the catalog into the Classifiers, the best scores must be determined. These scores will be used in the explanation phase.

The first model used is a **Graph Isomorphic Neural Network (GIN)**. This model is the first choice when conducting this research because of its unique ability to handle the node-order restriction in graph representations, as well as find the existence of specific subgraph patterns. The GIN results are not outstanding but are the first step in creating explanations on Android Malware CFG. The average results for classification using the GIN Classifier are shown in Table 1:

Neural Network Used	F1 Score	Accuracy
Graph Isomorphic Network	0.627	0.669

Table 1

Average F1 and Accuracy Scores for the GIN classification

The GIN Classifier can achieve only an average of 67% Accuracy and 63% F1 Score while classifying 3241 CFGs over 40 epochs. Each of the CFGs is batched into the model in groups of 20 graphs. Batch size is one of the most important hyper-parameters to tune in modern deep learning systems, but too large of a batch size will lead to poor generalization [10]. Through the process of tuning these parameters, it is found that CFG batches of 20 provided some of the best results, allowing for a balance between computational time and number of nodes and edges per classification. The GIN implemented 5 Multi-Layer Perceptron (MLP) and GIN layers, with 50 hidden layers. Many different layers are attempted to increase the F1 and Accuracy Scores but proved to perform the best with the aforementioned parameters. For instance, giving two examples, 3 MLP and GIN layers were attempted and received only a 64% Accuracy score, whereas 7 MLP and GIN layers received only 61% Accuracy. Changing the Hidden Layers has minimal effect on the scores over the epochs. Furthermore, 'neighbor pooling type' is set to 'sum' and receives the highest results out of the three parameters of 'sum', 'mean', and 'max'. The GIN is set to return the score over each layer to perform at the highest F1 and Accuracy Scores. During the training phase using the GIN, the CFG is classified on a few different techniques. The first technique put through the classifier is the 'Initial Access' Technique. This performs an output of 57% Accuracy and has an F1 score of 72%. The rest of the TTP are fed through the GIN, resulting in the average scores previously mentioned. Table 2 is a breakdown for each of the TTP passed through the Neural Network:

Graph Isomorphic Neural Network		
TTP	F1 Score	Accuracy
Initial Access	0.727	0.571
Execution	0.782	0.643
Defense Evaision	0.765	0.612
Credential Access	0.69712	0.714
Discovery	0.4	0.667
Lateral Movement	0.568	0.667
Collection	0.447	0.809
Average Scores:	0.627	0.669

Table 2

GIN Neural network classification results for each TTP

After receiving these results, a new architecture is used in order to find a solution that receives better scores. **A Graph Attention Network (GAT)** is a neural network architecture that operates on graph-structured data, leveraging masked self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations [7]. With this new architecture in use, new average results are given in Table 3.

Neural Network Used	F1 Score	Accuracy
Graph Attention Network	0.495	0.675

Table 3

Average F1 and Accuracy Scores for the GAT Classifier

The GAT Classifier is also given many trials with different parameters and hyper-parameters tweaked to adjust for the best Accuracy and F1 Score. The two main parameters that are adjusted to improve performance are the 'Hidden Dimensions' and the 'Number of Heads' parameters. With the GAT Classifier, each 'Head' specified will have a certain number of 'Hidden Dimensions' attached to it [8]. Ultimately, the 'Hidden Dimensions' and the 'Number of Heads' parameters make little difference in improving the model and do not make looking further into this implementation meaningful. Giving two examples, when the 'Number of Heads' are set to 40 and the 'Hidden Dimensions' are set to 20, the F1 and Accuracy Scores are 48.2% and 66.3% respectively. Giving the parameters a change to 'Number of Heads' set to 10 and the 'Hidden Dimensions' set to 50 does little to change these metrics. Another important consideration worth mentioning is that the GAT is computationally more efficient than the GIN approach allowing for 80 epochs to be used instead of the 40 used in the GIN Classifier. Even with this computational advantage, the GAT did not perform adequately enough to continue onto the explanation. A breakdown of the performance for each TTP is seen in Table 4.

Graph Attention Neural Network		
TTP	F1 Score	Accuracy
Initial Access	0.601	0.429
Execution	0.782	0.642
Defense Evaision	0.432	0.761
Credential Access	0.382	0.619
Discovery	0.458	0.846
Lateral Movement	0.323	0.476
Collection	0.487	0.952
Average Scores:	0.495	0.675

Table 4

GAT Neural Network Classification results for each TTP

Both the architectures of the Graph Isomorphic Neural Network and Graph Attention Neural Network returned similar Accuracy Scores, having an average Accuracy of around 67%. The F1 Scores, however, differed between the two architectures by almost 12%. To keep the computational cost down but still get accurate results of the classification for each TTP, the results of each specific TTP were completed on a subset of the entirety of the dataset.

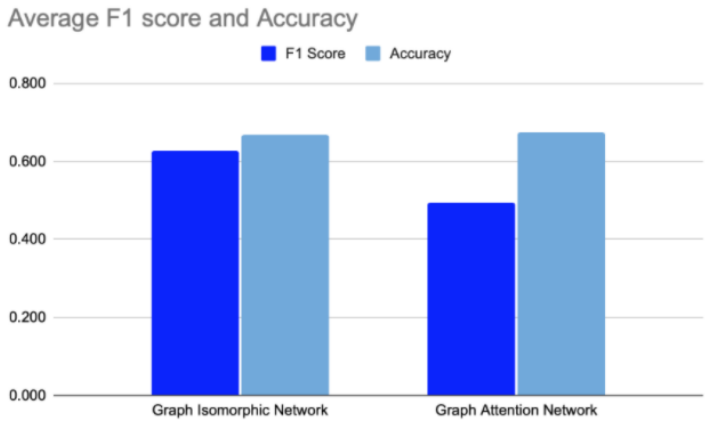

Figure 12

Average F1 Score and Accuracy for each Graph Neural Network

Furthermore, when conducting tests on each individual TTP (such as on the 'Lateral Movement' TTP, which has a frequency of 2.46%), many times the Neural Network Classifiers would fail to give Accuracy results above 50% or any relevant F1 score. These results are less than ideal and do not give confidence in returning accurate results when put through the subgraph explanation phase that is to follow; a new classification method is explored.

Upon completion of the implementation of the Neural Networks, another approach is explored. Utilizing the dataset that is collected through this research, a Decision Tree Ensemble learning approach is feasible. There are a couple of preliminary models that are put into use; Decision Tree, Random Forest, and Extra Trees Classifiers are tested. These classifiers are given by SciKit learn Machine Learning Library and adapted for use in this research [28].

The first updated Machine Learning approach is done using a single Decision Tree. This is done to explore if using Decision Tree Ensemble and Regression approaches are feasible in returning accurate results needed for the explanation phase. As this test is done to explore the validity of using this learning approach, the default parameters are used on the Decision Tree. After confirming the results of the Decision Tree, and getting promising F1 Scores and Accuracy ratings, a Random Forest and Extra Trees Classifier are used. Each of these utilizes 1000 trees in their implementation and sets 'Class Weight' as balanced. This provides better results, as seen in Table 5.:

Classifier	F1 score	Accuracy
Decision Tree	0.897	0.856
Extra Trees	0.92	0.89
Random Forest	0.927	0.896

Table 5

Average results for classification of each TTP

The results given for the Decision Tree, Random Forest, and Extra Trees Classifiers perform well on the dataset. The Decision Tree using the default arguments proved that this learning approach is a valid approach and gave way to the use of the Random Forest and Extra Trees Classifiers. An advantage to this implementation is the time component to training and implementing models. These Classifiers can read in all the necessary data, train and test the model in around 1/15 of the time that it takes to train and test the Neural Network approaches. Having confirmation that this approach is successful, each of the TTP is passed through the classifiers. Below, there is a breakdown for each TTP passed through each Classifier.

For this approach, unlike with the Graph Neural Networks, the full dataset is pushed through each TTP classification, as the computation time of the Decision Tree Ensemble is significantly faster. The values returned give the indication that using Decision Tree Ensemble Classifiers return much better results in classification and explanation than the Graph Neural Network approach.

Decision Tree Classifier		
TTP	F1 Score	Accuracy
Initial Access	0.86	0.83
Execution	0.86	0.8
Defense Evaision	0.92	0.86
Credential Access	0.88	0.86
Discovery	0.98	0.96
Lateral Movement	0.84	0.79
Collection	0.94	0.89
Average Scores:	0.897	0.856

Table 6

Decision Tree Classification results for each TTP

Following the use of the Decision Tree, a Random Forest is put into use. With the parameters tuned to 1000 Decision Trees, and a balanced class weight, the Accuracy Score increased by an average of about 4%, and the F1 Score increased by an average of about 3% from the single Decision Tree Classification.

Random Forest Classifier		
TTP	**F1 Score**	**Accuracy**
Initial Access	0.9	0.88
Execution	0.88	0.83
Defense Evaision	0.95	0.9
Credential Access	0.91	0.9
Discovery	0.99	0.97
Lateral Movement	0.89	0.85
Collection	0.97	0.94
Average Scores:	0.927	0.896

Table 7

Random Forest Classification results for each TTP

The last Decision Tree ensemble that is used is the SciKit Learn ExtraTreesClassifier. This Classifier is used because the computational cost and execution time is faster. This algorithm saves time because it randomly chooses the split point and does not calculate the optimal one. Because of this unique property, some aspects of the classification results may improve [19] [20]. In this situation, the F1 and Accuracy scores improved by 3% and 4% respectively from the single Decision Tree Classifier.

Extra Trees Classifier		
TTP	**F1 Score**	**Accuracy**
Initial Access	0.89	0.87
Execution	0.89	0.84
Defense Evaision	0.95	0.91
Credential Access	0.91	0.9
Discovery	0.99	0.97
Lateral Movement	0.89	0.85
Collection	0.97	0.95
Average Scores:	0.920	0.890

Table 8

ExtraTrees Classification results for each TTP

As was previously seen in Table 5, the distribution of each of the implemented Decision Tree Ensemble Classifiers is minimal. Being sure that no other method performs in a superior fashion, the last approach explored is the use of K-Nearest Neighbor (KNN) and Logistic Regression (LR) models. The KNN was used to estimate how likely the TTP is to be a member of one group or the other depending on what part of the CFG the TTP nearest to it are, whereas the Logistic Regression Classifier creates a model through regression analysis [23]. These two models were used to take a new approach separate from the Decision Tree Ensembles seen previously. After adapting the

models from SciKit Learn, these two models are put to test on the dataset and yield the following results:

Classifier	F1 score	Accuracy
K-Nearest Neighbor	0.916	0.886
Logistic Regression	0.881	0.829

Table 9

Average results for classification of each TTP

These results are like those seen in the Decision Tree Ensemble used previously. Each of the Graph Neural Network models begins classifying each TTP. The results are seen as follows:

K-Nearest Neighbor		
TTP	F1 Score	Accuracy
Initial Access	0.88	0.86
Execution	0.88	0.82
Defense Evaision	0.94	0.89
Credential Access	0.89	0.88
Discovery	0.98	0.97
Lateral Movement	0.88	0.85
Collection	0.96	0.93
Average Scores:	0.916	0.886

Table 10

K-Nearest Neighbor Classification results for each TTP

After implementing the K-Nearest Neighbor Classifier, the results are on par with what is seen with the Decision Tree Classifiers, but still not surpassing the results given by the Random Forest or ExtraTrees Classifier. The next Classifier, the Logistic Regression model, is now used for all the TTP.

Logistic Regression		
TTP	F1 Score	Accuracy
Initial Access	0.81	0.76
Execution	0.84	0.75
Defense Evaision	0.93	0.87
Credential Access	0.84	0.81
Discovery	0.99	0.97
Lateral Movement	0.82	0.75
Collection	0.94	0.89
Average Scores:	0.881	0.829

Table 11

Logistic Regression results for each TTP

For each of the Decision Tree Ensemble, KNN, and Regression Classifiers the averages tend to stay around the same values. This gives confidence that the data collected is adequate for an explanation following the classifications.

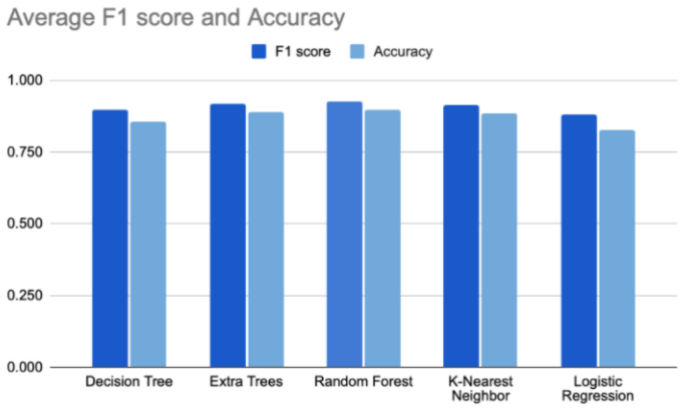

Figure 13

Average F1 and Accuracy Scores Decision Tree Ensemble, KNN and Logistic regression models

The results of the Random Forest Classifier and the ExtraTrees Classifier are closely related. The decision to move forward with the Explanation is between these two Classifiers. ExtraTrees Classifier is an excellent choice because it is computationally faster allowing for increased explanation performance. Considering all the Decision Tree Ensemble, KNN, and Logistic Regression Classifiers, the average F1 Score and Accuracy far surpass that of the Graph Neural Network Classifiers.

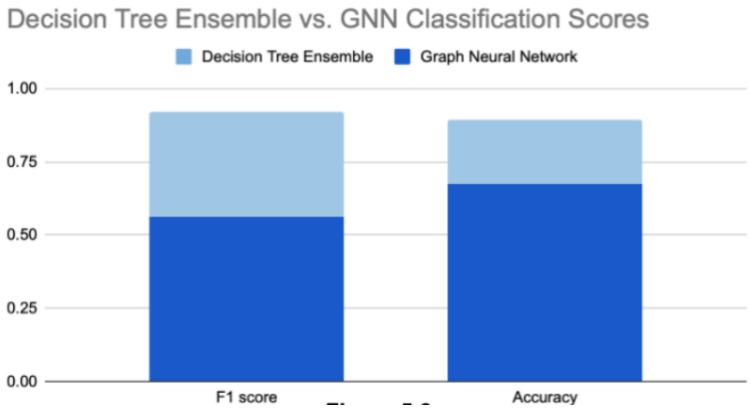

Figure 14

Average F1 and Accuracy Scores for graph Neural Network vs. Decision Tree Ensemble

With an overall average F1 Score of 90.8% and a returned accuracy of 87.1% given for each of the Decision Tree Ensemble, KNN and Logistic Regression Classifiers used, these scores give confirmation that explanations of the graphs are possible and feasible given the constraints. The scores in this average do allow for an effective explanation of the subgraphs.

The Random Forest and ExtraTrees Classifiers especially yield useful and accurate information superior to that of the Graph Neural Networks; thus, experimentation and explanation is pursued on these Classifiers, and specifically on the ExtraTrees Classifier. The depth of correct F1 and Accuracy Scores can be seen in the following Matrix.

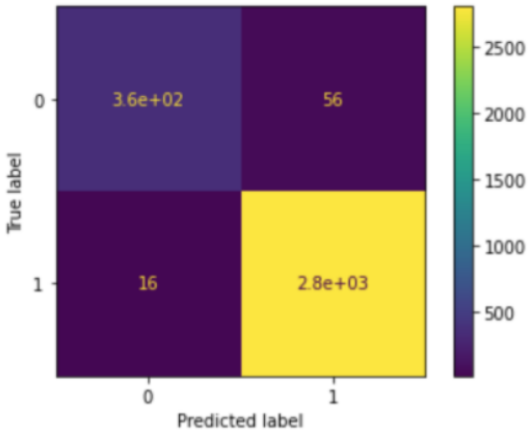

Figure 15

This matrix demonstrates the metrics given by the ExtraTrees Classification

This Matrix demonstrates how well the model did on the dataset. In this case, the model used was the ExtraTrees Classifier, as this classifier will be used for the explanation to follow. Experiencing scores in this range is crucial to have effective and accurate explanations.

8 Graph Explainability

The interpretation and explainability of the machine learning model are necessary for the confirmation of results. The tool SHapley Additive exPlanations (SHAP) is used to explain the machine learning's predictions. The features of the model are assigned a value of importance for a certain prediction [11].

After implementing and saving a model for each of the TTP classified by the ExtraTrees Classifier, the SHAP values are collected. The SHAP Values are an explanation of the model's decisions for each of the TTP. These SHAP values are then used to calculate the edge importance for every edge within the Control Flow Graph. For each of these edges, an edge value has been assigned by using the implementation given by Joaristi, M., & Serra, E. (2021), in their paper *SIR-GN: A Fast Structural Iterative Representation Learning Approach For Graph Nodes* [9]. By taking the dot product of the SHAP Values and the edge values, the edge importance has been calculated for each edge of the Control Flow Graph. The Edge Importance is sent to a pickle file containing each specific edge in the CFG and the Edge Importance for each of the TTP. The Edge Importance is then able to explain in which subgraph the predicted TTP can be found.

Graph data are not naturally processed through standard machine learning models. This research employs graph representation learning such as SIR-GN which produces a vectorial representation for each node. Given the vectorial representation of SIR-GN, it provides a procedure to create a

unique graph representation. Performing qualitative analysis validates that all the API calls for each subgraph responsible for a TTP classification are related. This analysis shows that the API calls selected by this method are always logically related to the TTP definition.

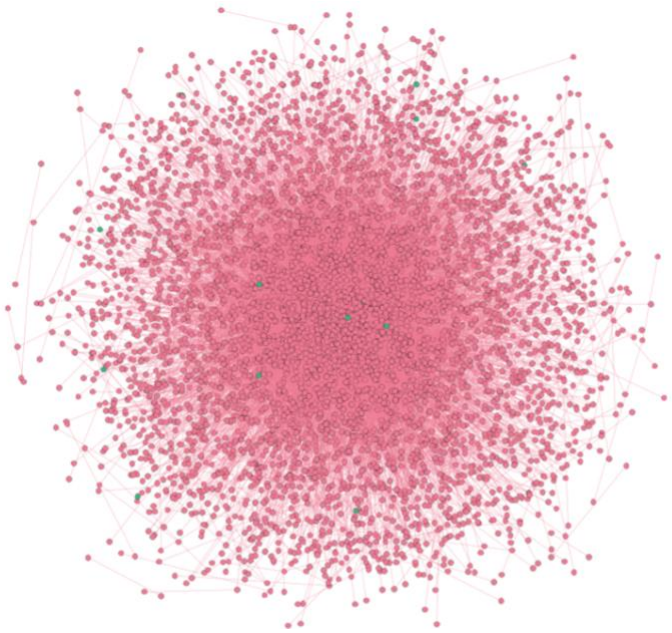

Figure 16

Control Flow graph utilizing TTP "Initial Access"

Above is a CFG of a malware executable utilizing the TTP “Initial Access”, for the malware with the SHA-256 hash 006c24ff3ea7248f01d615d882eb993b88e096772bfeb6840c0cdc 5527e0d97d.

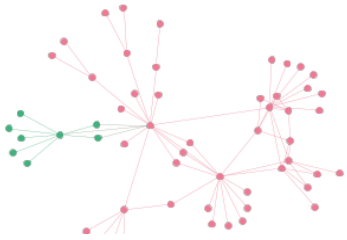

Figure 17

CFG subgraph where the TTP "Initial Access"

is specifically being used

Figure 8.2 shows the subgraph identification for the malware with the given SHA-256 value seen above. This demonstrates the subgraph where the TTP "Initial Access" was found inside of the entirety of the Control Flow Graph.

9 Data Statistics

Several observations have been made through the analysis of the collected malware dataset. From the 3244 values of nodes and edges, the values most often seen are 60 nodes and 73 edges, which always occur together in this dataset. These values occur 1385 times within the malware samples. The minimum number of nodes and edges are the same in the dataset at 0, and the maximum number of nodes is 136993 and the maximum number of edges is 333854. These maximum values occur together in the collected malware samples. The average number of nodes in the dataset is 5775.393, while the average number of edges is 12581.291. The growth rate of this dataset is graphed below.

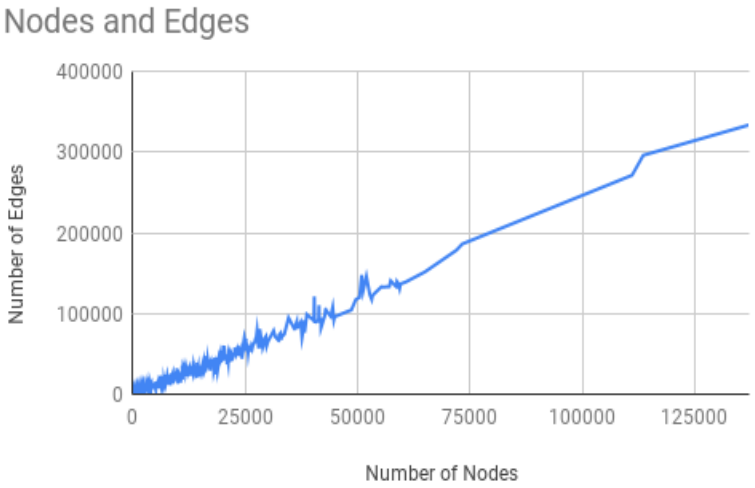

Figure 18

The growth rate of nodes and edges seen within the CFG dataset

Through the collection of malware samples from VirusTotal, the frequency at which the TTP occur throughout the entire data set is found. There are certain TTP that are more efficient to use to train the machine learning classifier. The closer the tactic is to 50%, the better the results from the classifier because the data is balanced. Accurate results can still be achieved with data occurring around 80%, but there are TTP that do not come close to 50%, such as "Privilege Escalation" and "Command and Control". These tactics are omitted as they had less than 1% of overall use within the dataset. The graph below shows the seven TTP that were used in the classification and their percentage of use within the dataset. The frequency at which the malware tactics occur in a dataset is graphed in Figure 19.

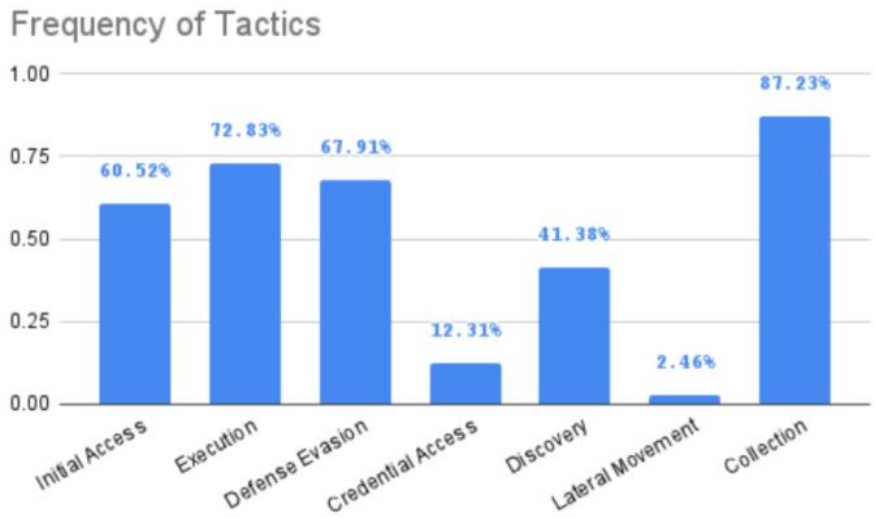

Figure 19

The figure shows the frequency of each individual TTP

Conclusion

The development of cybersecurity as a necessary field of study comes from the increasingly intertwined relationship people have with technology. One branch of cybersecurity is malware analysis, where one deciphers the technical jargon and behaviors of malware and develops it into something digestible for malware analysts. The identification of subgraphs in the Control Flow Graph of malware demonstrating the use of a specific TTP is an important task that facilitates the understanding of malware behavior and then its mitigation. Given the experimental results, this novel methodology of SIR-GN Classification outperforms, in terms of accuracy and F1-score, the capability of graph isomorphic networks and graph attention neural networks, and it provides CFG subgraphs that effectively characterize the malware behavior. This research provides an automation methodology to locate TTP in a sub-part of the control flow graph with about 89% accuracy and 92% F1 score.

Future Work

This research has currently been accepted for presentation and publication at the SigmaXi, AAAI, and IEEE BigData Conferences. Continued work is being performed, and more data is being passed through the SIR-GN classifiers to achieve greater scores. However, an additional method that can be used to classify malware is by using small, induced subgraphs called graphlets. Graphlets can be mapped onto a CFG by using their orbits, and then the malware can be detected and assigned to a specific malware family. One complication regarding graphlets is they are most efficient when they are small, essentially five or fewer nodes. CFG's can be rather large, so this creates a problem. Potentially, one can calculate the treewidth of the graph and use this as a bound for the graphlets to mitigate this issue.

References

[1] A. Georgiadou, S. Mouzakitis, and D. Askounis, "Assessing mitre att&;ck risk using a cyber-security culture framework," Sensors, vol. 21, no. 9, p. 3267, 2021.

[2] A. Junction, B. E. Strom, J. A. Battaglia, M. S. Kemmerer, W. Kupersanin, D. P. Miller, C. Wampler, S. M. Whitley, and R. D. Wolf, "Finding Cyber Threats with ATT&CK™-Based Analytics," Jun. 2017.

[3] A. Kapoor and S. Dhavale, "Control flow graph based multiclass malware detection using bi-normal separation," Defence Science Journal, vol. 66, no. 2, p. 138, 2016.

[4] "Class index : Android developers," Android Developers. [Online]. Available: https://developer.android.com/reference/classes. [Accessed: 27-Jul-2021].

[5] "Free automated malware analysis service - powered by Falcon Sandbox," Hybrid-Analysis. [Online]. Available: https://www.hybrid-analysis.com/. [Accessed: 27-Jul-2021].

[6] G. Ayoade, S. Chandra, L. Khan, K. Hamlen, and B. Thuraisingham, "Automated threat report classification OVER Multi-Source Data," 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Nov. 2018.

[7] "Graph Attention Network," GAT Explained | Papers With Code. [Online]. Available: https://paperswithcode.com/method/gat. [Accessed: 27-Jul-2021].

[8] H. Zhang, M. Li, M. Wang, and Z. Zhang, "Understand Graph Attention Network," DGL, 2018. [Online]. Available: https://docs.dgl.ai/en/0.6.x/tutorials/models/1_gnn/9_gat.html. [Accessed: 27-Jul-2021].

[9] Joaristi, M., & Serra, E. (2021). SIR-GN: A Fast Structural Iterative Representation Learning Approach For Graph Nodes. ACM Transactions on Knowledge Discovery from Data (TKDD), 15(6), 1-39.

[10] K. Shen, "Effect of batch size on training dynamics," Medium, 19-Jun-2018. [Online]. Available: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e. [Accessed: 27-Jul-2021].

[11] Lee, S & Lundberg, S (2017). A Unified Approach to Interpreting Model Predictions. CoRR

[12] M. Eskandari and S. Hashemi, "Metamorphic Malware Detection using Control Flow Graph Mining," IJCSNS International Journal of Computer Science and Network Security, vol. 11, no. 12, Dec. 2011.

[13] M. Hassen and P. K. Chan, "Scalable function call graph-based malware classification," Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, 2017.

[14] MITRE ATT&CK®. [Online]. Available: https://attack.mitre.org/. [Accessed: 27-Jul-2021].

[15] "Mobile Matrices," MITRE ATT&CK®, 09-Feb-2021. [Online]. Available: https://attack.mitre.org/matrices/mobile/. [Accessed: 30-Jul-2021].

[16] "Package index : Android developers," Android Developers. [Online]. Available: https://developer.android.com/reference/packages. [Accessed: 27-Jul-2021].

[17] P. Xu, C. Eckert, and A. Zarras, "Detecting and categorizing Android malware with graph neural networks," Proceedings of the 36th Annual ACM Symposium on Applied Computing, 2021.

[18] PyTorch. [Online]. Available: https://pytorch.org/. [Accessed: 27-Jul-2021].

[19] P. Aznar, "What is the difference between Extra Trees and Random Forest?," QuantDare, 17-Jun-2020. [Online]. Available: https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/. [Accessed: 27-Jul-2021].

[20] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," Machine Learning, vol. 63, no. 1, pp. 3–42, 2006.

[21] "sklearn.ensemble.ExtraTreesClassifier," Scikit. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html. [Accessed: 27-Jul-2021].

[22] S. S. Anju, P. Harmya, N. Jagadeesh, and R. Darsana, "Malware detection using assembly code and control flow graph optimization," Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India - A2CWiC '10, no. 65, pp. 1–4, Sep. 2010.

[23] Techopedia, "K-Nearest Neighbor (K-NN)?," Techopedia, 14-Mar-2017. [Online]. Available: https://www.techopedia.com/definition/32066/k-nearest-neighbor-k-nn. [Accessed: 28-Jul-2021].

[24] VirusTotal. [Online]. Available: https://www.virustotal.com/gui/. [Accessed: 27-Jul-2021].

[25] "Welcome to Androguard's Documentation!," Androguard. [Online]. Available: https://androguard.readthedocs.io/en/latest/. [Accessed: 27-Jul-2021].

[26] W. Hu, "How Powerful are Graph Neural Networks?," GitHub. [Online]. Available: https://github.com/weihua916/powerful-gnns. [Accessed: 27-Jul-2021].

[27] X. Hu, T.-cker Chiueh, and K. G. Shin, "Large-scale malware indexing using function-call graphs," Proceedings of the 16th ACM conference on Computer and communications security - CCS '09, Nov. 2009.

[28] "1. Supervised learning," Scikit. [Online]. Available: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning. [Accessed: 27-Jul-2021].

[29] "Angr." *Angr*, http://angr.io/.

[30] *VX Vault*, http://vxvault.net/ViriList.php.

[31] "Automated Malware Analysis Solutions." *Hatching*, https://hatching.io/.

[32] Rizvi, S. K. J., Aslam, W., Shahzad, M., Saleem, S., & Fraz, M. M. (2021, October 12). *Proud-mal: Static analysis-based progressive framework for deep unsupervised malware classification of Windows Portable executable*. Complex & Intelligent Systems. Retrieved November 5, 2021, from https://link.springer.com/article/10.1007/s40747-021-00560-1.