Machine Learning Optimization for Enhanced OS Fingerprinting

Spencer Ekeroth
Virginia Tech

Jeremy Neale
Virginia Tech

Jae Sung Kim
Virginia Tech

Abstract

Operating System (OS) Fingerprinting is a technique that can be used to identify a networks operating systems by evaluating network traffic in the form of TCP/IP packets. This can give a black-hat threat actor valuable information to help map out a network's architecture, as well as exploit pre-existing vulnerabilities on targeted operating systems. This research will explore the effectiveness of different machine learning models on the CIDS-2017 dataset, a collection of over 47 gigabytes of pcap files with their corresponding operating systems.

This research also proposes a new command line interface, OsirisML, which uses *nPrint*[7] to preprocess the data into tabular data and *XGBoost*[9] to apply ML to the data to generate, retrain, and test ML models. Using this tool, OS Fingerprinting models have been generated with high success metrics.

1 Introduction

Passive fingerprinting analyzes network traffic offline, which avoids detection from the network. There are many versions of passive fingerprinting tools that can analyze packet capture (*pcap*) data and attempt to identify the operating system. To generate a new or custom passive fingerprinting model using machine learning, *pcap* data can be analyzed and used to generate a new model with a tool like *nPrintML*, which uses *nPrint* to transform the PCAP data into tabular data, and *AutoML* [5] automatically generate a ML model from the tabular data.

Given the dynamic nature of network environments, where new OS versions are continuously released, there is a compelling need for tools that can adapt and retrain machine learning models for passive OS fingerprinting to remain effective against the ever-evolving landscape of operating systems.

This research introduces a novel command-line tool, OsirisML, which consists of two main components similar to *nPrintML*. First, it leverages *nPrint* to transform network traffic (pcap files) into a normalized tabular format suitable for machine learning. Unlike *nPrint*s alternative, *nPrintML*, that utilizes *AutoML* for automated model generation, OsirisML integrates the advanced capabilities of XGBoost—an algorithm renowned for its efficiency with tabular data—to enhance model accuracy and adaptability. The usage of XGBoost [9] for machine learning applications contains 4 components: generating models, testing unseen data with a model (passive fingerprinting), retraining a model on an unseen dataset to generate a new model, and hyperparameter tuning (section 9).

This tool allows for 3 significant advantages over its counterparts:

- Hands-on manipulation of the model creation with custom hyperparameter tuning, file partitioning, and retraining models
- Improved accuracy and F-1 scores from model generation on identical data
- More efficient RAM usage on identical data

2 Motivation and problem statement

As aspiring cybersecurity professionals, it is understood that the importance of identifying an operating system over a network can be of critical importance. Creating an accurate tool to do so would be beneficial for independent work, as well as to the global community of white hat threat actors. It could be used to impersonate a black-hat threat actor to see if they could enumerate a network and exploit any vulnerabilities.

Open source work is what has propelled the computer science field forward for the past century. Projects such as Linux, GNU, Python, and Docker have benefited millions of people across the world. Improving upon pre-existing open source tools would be one way to advance the fields of cybersecurity and machine learning and help other computer scientists in the future.

The goal of OsirisML is not just to make an all-in-one passive fingerprinting model, but to provide a method to create and build upon pre-existing models. This allows OS Fingerprinting to stay up to date and continuously improve from open-source contributions of datasets and model generation.

3 Related work

In the research paper, 'New Directions in Automated Traffic Analysis' [3], Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal created a tool called *nPrint* and *nPrintML* [7]. *nPrint* is capable of capturing traffic live or processing a pcap file to convert raw packet data into a tabular format suitable for machine learning algorithms. *nPrintML* is an all inclusive tool that takes the tabular output of *nPrint* and uses *AutoML* to select a machine learning algorithm to create an optimized model for OS fingerprinting. As more machine learning models are created and better optimized to handle tabular data, this project will focus on optimizing the *nPrint* data output and using modern machine learning algorithms to attain a model that is more accurate in passive OS fingerprinting than what was achieved by Jordan Holland et al. [3]

4 Production Environment for OsirisML

The production environment for OsirisML is set up on a virtual machine running Debian Linux, specifically Ubuntu 22.04 [8], to ensure stability and consistency. The virtual machine is running on an AMD EPYC 7401 24-Core Processor with 128GB of RAM. Virginia Tech's rlogin cluster, which is a multi-machine system running Linux to provide computational support for Computer Science instruction, was also utilized to run *hyperparameter_tuning.py*. The rlogin cluster has many nodes, each equipped with 2 Xeon Gold 5218 CPUs with 2.3Ghz 16 core processors, 384 of RAM, and a 10 Gbit Ethernet connection [6].

To use *nPrintML* in the same environment as OsirisML, utilize *pyenv* [1] to manage Python versions, specifically Python 3.8 to accommodate the different Python version requirements of *nPrintML* (Python 3.8) and OsirisML (Python 3.12). This approach allows both tools to run smoothly and concurrently, ensuring compatibility and optimal performance for each.

To configure the environment, *configure/configure.sh* installs all dependencies, including the installation of *nPrint* 1.2.1 via tarball extraction. This script installs the correct versions of all required libraries using *apt* and *pip*. Despite the limitations imposed by *nPrintML*, OsirisML allows the latest versions of Python and other libraries, leveraging the newest features and improvements in the ecosystem. Python 3.12 introduces performance enhancements in reducing memory consumption, which is crucial for efficiently processing the large *pcap* files.

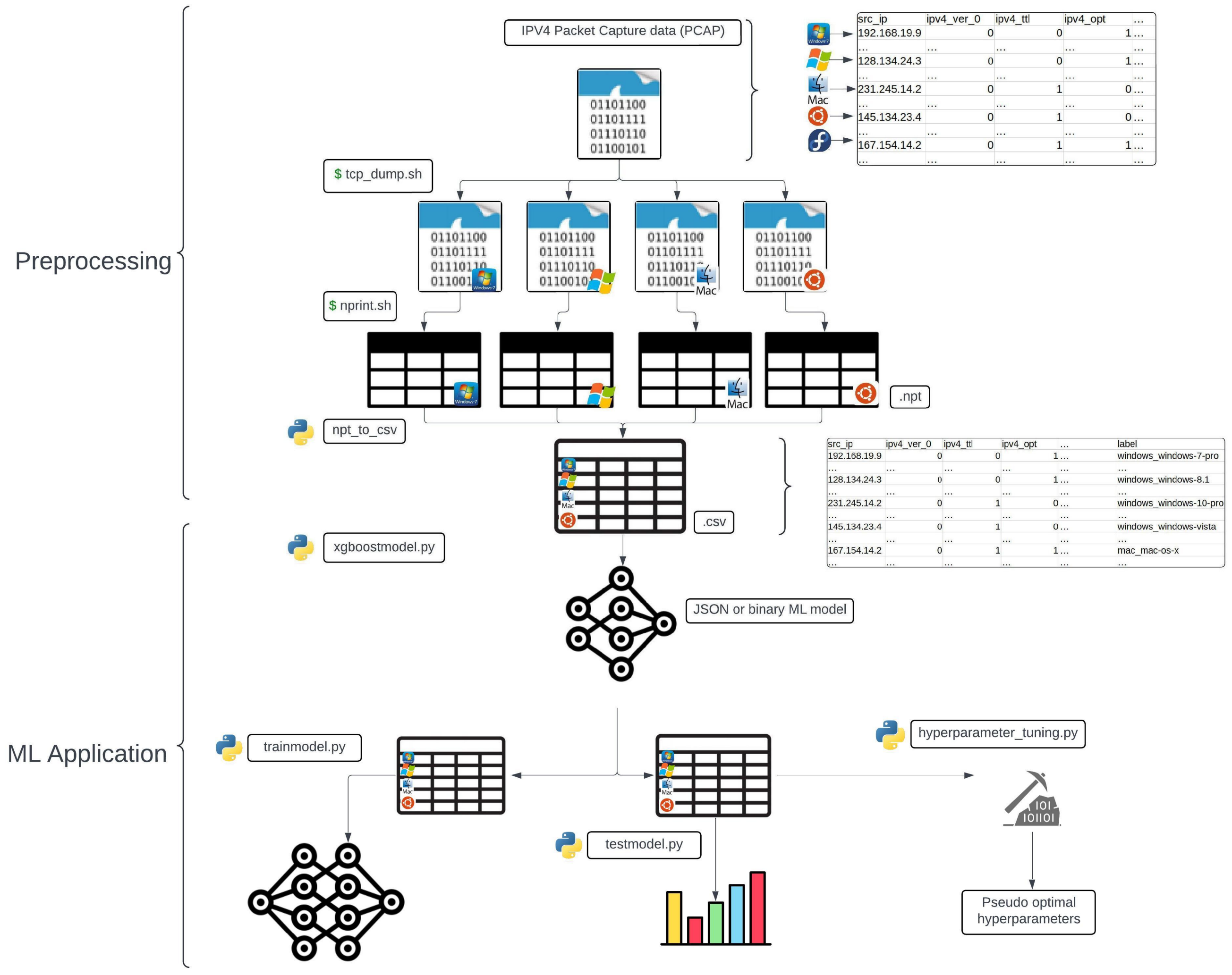
5 CIC-IDS 2017 PCAP Dataset

The CIC-IDS 2017 dataset [4], released by the University of New Brunswick, is a comprehensive collection of network traffic designed to represent a mix of the typical network traffic in a real-world environment, along with intrusion attempts from Kali Linux. This dataset is chosen because it contains a large set of labeled operating systems by IP address. The networks are represented by an array of operating systems, including different versions of Ubuntu servers, Windows machines from Vista to Windows 10, and macOS systems, among others. Operating system fingerprinting often relies on discerning the subtleties in packet structure, sequence numbers, fragments, window sizes, and other TCP/IP stack behaviors that vary from one OS to another.

6 nPrint and nPrintML

The starting point for developing a model that is capable of differentiating fine-grain operating systems was to replicate Section 5.2 in 'New Directions in Automated Traffic Analysis' [3]. By using the same features, it would provide a benchmark for the new model. *nPrint* takes a unique approach to packet representation by assuming all packets have a header of maximum length. Any missing options or fields in the headers are represented with a value of -1. To keep consistency with Section 5.2 only IPv4 and TCP headers will be extracted from each packet. This results in each encoded packet sample to have 960 features, representing the 480 bits from the IPv4 header and the additional 480 bits from the TCP header.

The output generated from *nPrint* with the IPv4 and TCP header options set still requires cleaning the IP source address, IP destination address, IP identification, TCP source port, TCP destination port, and both TCP sequence and ack number, which can act as direct identifiers to the operating system. This cleaning will result in each sample in the dataset to have less than 960 features. These changes can be made to the source code of *nPrintML*. After using pip to install the *nPrintML* package, it is imperative to edit *automl.py* which can be found in *site-packages/nprintml/learn/* where site-packages is the directory pip installs new libraries and packages. Before the *train* method is called, the areas of the IPv4 and TCP header mentioned above need to be dropped as they act as direct identifiers. If the areas are not dropped, *nPrintML* will return a model that is extremely overfitted to the dataset due to data leakage.



src_ip	ipv4_ver_0	ipv4_ttl	ipv4_opt	…
192.168.19.9	0	0	1	…
…	…	…	…	…
128.134.24.3	0	0	1	…
…	…	…	…	…
231.245.14.2	0	1	0	…
…	…	…	…	…
145.134.23.4	0	1	0	…
…	…	…	…	…
167.154.14.2	0	1	1	…
…	…	…	…	…

src_ip	ipv4_ver_0	ipv4_ttl	ipv4_opt	…	label
192.168.19.9	0	0	1	…	windows_windows-7-pro
…	…	…	…	…	…
128.134.24.3	0	0	1	…	windows_windows-8.1
…	…	…	…	…	…
231.245.14.2	0	1	0	…	windows_windows-10-pro
…	…	…	…	…	…
145.134.23.4	0	1	0	…	windows_windows-vista
…	…	…	…	…	…
167.154.14.2	0	1	1	…	mac_mac-os-x
…	…	…	…	…	…

Figure 1: Visualization for the flow of both phases of the CLI

7 Phase 1 - Data Preprocessing Workflow

OsirisML is split into two main components: preprocessing, and ML application. The goal of the preprocessing is to take *pcap* data and convert it to tabular (*csv*) data, making it compatible for machine learning algorithms. This entire process can be executed with `./process_pcap.sh`, which calls the following three scrips found in *preprocessing/* in succession:

1. **tcp_dump.sh** - uses the source IP labels provided on CIC-IDS2017 host website [4] by default to split up traffic based on operating systems. Alter the source code of this file with new source IPs if attempting to generate a model from a dataset with different source IPs.

   Note that the PCAP file that this script accepts as a system argument **must** be located in *data/pcap*, and the resulting PCAP files will be placed into *data/pcap/pcap_os_split/*. The name of each file will be the name of the operating system provided in the source code.

   ```
   Usage: ./tcp_dump.sh <PCAP_file_name.pcap>
   ```

2. **nprint.sh** - unlike *nPrintML*, *nPrint* is unable to take a directory of pcaps as a valid argument. This script loops through all PCAP files in *data/pcap/pcap_os_split/* and calls *nPrint* on each file with -4 -t parameters - see *nPrint* documentation.

   The resulting NPT files will be placed into *data/npt/* and will have the same names with new extensions.

   ```
   Usage: ./nprint.sh
   ```

3. **npt_to_csv.py** - This file loops through each NPT file in *data/npt/* and writes each line into a single CSV file, which it places in *data/csv/*.

   ```
   Usage: python3 <npt_to_csv.py> <new.csv>
   ```

Execute this entire workflow with:

```
./process_pcap.sh <PCAP_file_name.pcap>
```

Note that process_pcap.sh also deletes the *data/npt/* and *data/pcap/pcap_os_split/* directories after finishing.

8 Phase 2 - ML Application with XGBoost

Once the tabular data is preprocessed, it has the relevant TPC/IP binary input attributes and one classification label, which is the corresponding operating system. With the labeled CSV file, different ML algorithms can easily be applied to the data to produce a model. XGBoost was selected for its effectiveness with tabular data, along with its open-source nature and extensive documentation.

The XGBoost scripts splits the CSV file into *x_train, x_test, y_train, and y_test*. The *x* files are the input attributes, which in this case are the TCP/IP binary input attributes, and the *y* files are the corresponding classification labels, which is the last column of the CSV. The labeled csv dataset has 962 columns (or attributes), but the first column is the source IP address in semantic form, and the last column is the classification label as a string.

The x_train and x_test values were then further cleaned to remove the bit ranges of values that would act as direct identifiers to the model. Both y_train and y_test were encoded from strings to integers to allow them to be processed by XGBoost's classifier.

The default eval metric supplied to all three model scripts is *mlogloss*, which is selected due to its nature of multi-class classification, and its robustness to imbalanced classes.

There are 4 main files in *model/*:

1. **hyperparameter_tuning.py** - see section 9.

   ```
   Usage: python3 hyperparameter_tuning.py
   <data.csv>
   ```

2. **xgboostmodel.py** - generates a model from the supplied *csv*. This uses a test size of 0.2 by default, but can be updated with an optional sys argument.

   The *csv* file must be located in *data/csv/*, where it should be by default after data preprocessing. The outputted *json* file will be placed into *model/json/* by default.

   ```
   Usage: python3 xgboostmodel.py <data.csv>
   <modelname> <OPTIONAL: <decimal for test
   split size>
   ```

3. **trainmodel.py** - *XGBoost* allows for already generated models to be retrained on new data to produce a new model. This is useful for optimized RAM and storage, as well as improving the robustness of a model. This script uses a *DMatrix*, where the *x* and *y* train and assigned to *dtrain*, and *x* and *y* test are assigned to *dval*. This matrix approach during training drastically improves space and speed.

   When invoking *xgboost.retrain*, the max number of boost rounds is set to 1000 and early stopping rounds is set to 10. Additional hyperparameter tuning is required to find a balance between model complexity vs. overfitting.

   Currently, max_depth = 10 is the only parameter supplied to this script, but this can be changed in the source code or may be updated in the future as hyperparameter tuning continues.

   The model must already be located in *data/json/*, which is should be after invoking xgboostmodel.py or this script. The training data must be located in *data/csv/*. The new model will be placed into *data/json/*.

   ```
   Usage: python3 trainmodel.py
   <current_model.json> <data.csv> <new.json>
   ```

4. **testmodel.py** - this script can be thought of as passive OS Fingerprinting. It takes a model generated by xgboostmodel.py or trainmodel.py, and tests unseen data to evaluate its performance. This script currently only returns the accuracy and F-1 score to evaluate the success of the model, but does not provide information on each individual machine in the packet capture data. To determine predictions for individual machine's operating system, supply only 1 source IP address in the *csv* file to the model.

   The model must already be located in *data/json/*, which is should be after scripts 2 and 3. The testing data must be located in *data/csv/*.

   ```
   Usage: python3 testmodel.py <model.json>
   <data.csv>
   ```

9 Hyperparameter tuning and optimization

In machine learning, *parameters* are learned and updated throughout the training process, while *hyperparameters* are set beforehand and determine key features of the training method such as model architecture, learning rate, and model complexity. *XGBoost* uses a gradient boosted decision tree machine learning algorithm that accepts many optional hyperparameters. Using optimal hyperparameters for the machine learning training process is crucial to achieving higher model accuracy and F-1 scores while avoiding overfitting. Hyperparameters also have a significant role in the time and computer resources necessary to train a model.

OsirisML comes equipped with a script *preprocessing/hyperparameter_tuning.py* that searches for the optimal parameters for the supplied CSV file, which is the PCAP data that has been preprocessed into tabular data. See section 8 for usage.

To determine the optimal parameters, the script uses *GridSearchCV* [2], which generates an XGBoost model for every combination of hyperparameters supplied as a parameter of a Python Dictionary, where the keys are the hyperparameter options as strings, and the values are the options for that hyperparameter. For example,

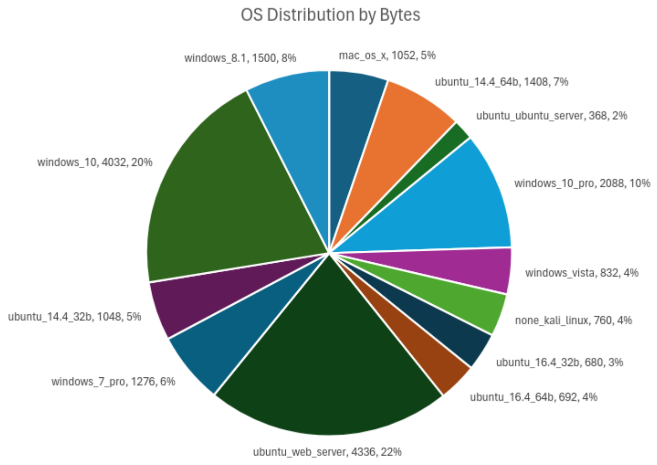
```
# Create the parameter grid
param_grid =
{
    'max_depth': [3, 6, None],
    'gamma': [0.1, 0.5, 1.0],
    'min_child_weight': [2, None]
}
```

Using GridSeachCV with this grid would generate 18 models, since there are $3 * 3 * 2$ combinations of the three parameters. After generating the 18 models, GridSeachCV returns the best combination of parameters based on the supplied evaluation metric, which is supplied as *mlogloss* by default in the script.

It is recommended to use a smaller partition of the data to search for hyperparameters, then to use these psuedo-optimal hyperparameters on the entire dataset. This is because searching for hyperparameters is computationally expensive and costs lots of time and computer resources. The *pcap* files can be partitioned with `python3 preprocessing/shrink_pcap.py <path_to_pcap.pcap> --fraction X (default 10).`

After retrieving the optimal combination of supplied hyperparameters, they can be supplied to *model/xgboostmodel.py* and used to train the entire data set.

Using the *Friday-WorkingHours.pcap* file from the section 5 dataset, a partition of the dataset was created that split the entire 8.2 gb datset 32 times. Windows 10 and the Ubuntu web server were both split 50 times, since they were drastically over represented compared to the other OSes. The resulting dataset had 134,270 samples and the *csv* file was now about 336 mb. The OSes were distributed by the following chart:



With this smaller dataset, the hyperparameter tuning script reported that max_depth = 10 and the others as *None* was optimal, yielding the highest accuracy and F-1 scores of 97.70% and 97.51% respectively. There are many more hyperparameters to test, but due to the nature of the time it takes to generate models, only a select pool of hyperparameters were given to *GridSearchCV*:

```
# 54 combinations - 3 * 3 * 2 * 3
param_grid =
{
    'n_estimators': [400, 800, None],
    'max_depth': [6, 10, None],
    'min_child_weight': [2, None]
    'gamma': [0.1, 0.5, 1.0]
}
```

Further research could test more hyperparameters and/or use a larger dataset to return a more accurate assessment of the optmial hyperparameters to generate models with.

10 Results

As previously mentioned in section 9, using a condensed version of *Friday-WorkingHours.pcap* from the CIC-IDS 2017 dataset produced an accuracy of 97.70% and an F-1 score of 97.51%, while *nPrintML* achieved 96% accuracy and 96% F-1 score (*nPrintML* only reports results with two digits). Using XGBoost's default parameters on the entire Friday file resulted in 74.09% accuracy and 64.83% F-1 score. When applying the optimal parameters recieved from the hyperparameter tuning to the same Friday file, the accuracy and F-1 score improved to 84.91% and 82.96% respectively. When using *nPrintML* on the same dataset, the process killed due to RAM usage.

The entire Friday file that received these metrics is originally an 8.2 gb *pcap* file that was preprocessed into a 11.2 gb *csv* file. It contains 4,501,930 samples, and *XGBoost* used the default 80% training and 20% testing split.

The final test was to replicate Section 5.2 of Jordan Holland et al. [3]. The first 100,000 packets of each operating system in *Friday-WorkingHours.pcap* made up the dataset that the model was to be trained and tested on. As there are only 13 operating systems, the dataset is comprised of 1,300,000 packets. Using this dataset with *XGBoost*'s optimal parameters saw a 90.11% accuracy and a 91.39% macro F-1 score.

11 Proposed work

Looking ahead with OsirisML, the plan is to train the current model on the rest of the *pcap* files within the CIC-IDS 2017 dataset, as it has only been trained on 1 out of the 5 available *pcap* files. This would ensure a comprehensive and robust final model.

Once the model has been trained on the complete CIC-IDS 2017 dataset, a comprehensive model will be ready to receive new training data from other sources. OsirisML is an open-source project, and users will then be able to contribute to the model by providing new *pcap* data, if approved by the OsirisML team. A new dataset could be produced by isolating a network and populating it with machines with different operating systems, which would allow complete control over the collection of data.

Another direction will involve extensive hyperparameter tuning to optimize the model's performance and accuracy as more hyperparameter types and options are introduced to *GridSearchCV*. Given more time and computing resources, the model can identify more specific hyperparameter combinations to optimize its accuracy and F-1 scores.

12 Potential Limitations

While OsirisML has seen great initial success, there are some factors that could have contributed to this, and other limitations regarding the tool as a whole. OsirisML has only been tested on the CIC-IDS2017 dataset which only consists of 13 operating systems. In order to be confident in the success of this CLI, its success would need to be replicated with other datasets.

Current compatibility limitations are that OsirisML is currently only available for installation on Debian Linux. This is because it is recommended to use this tool in a VM, which is likely GNU-Linux based. Containerizing could make the tool more accessible and consistent across host machines.

13 Statement of work

Spencer Ekeroth - Wrote xgboostmodel.py and train-model.py to create a model and receive first accuracy and f1 scores. Researched and tested *nPrint* data representation to understand feature set and how to clean data of direct identifiers. Wrote pcap_to_nprint.sh script to automate pcap conversions with *nPrint*. Wrote npt_to_csv.py with Jeremy to create fully labeled dataset to be used with xgboostmodel.py. Edited nPrintML source code to be able to get accurate results to compare against our own model. Wrote helper scripts to set up training described in section 5.2 and then went on to test our model with the section 5.2 restrictions. Created HuggingFace organization and have updated datasets accordingly. Wrote section 6 and 10 of this paper.

Jeremy Neale - Created all of tcp_dump.sh, process_pcap.sh, shrink_pcap.py, hyperparameter_tuning.py, configure_.sh, and assisted with many of the other scripts, notably npt_to_csv.py (created by Spencer). Configured VM and documented installation process to create entire configure directory. Split Friday 32 times and ran the hyperparameter tuning script with nohup to retrieve optimal parameters. Ran entire workflow on Thursday and Friday PCAP files on VM to generate results. Created pie chart, workflow diagram (with Lucid), and github README. Wrote sections 0, 1, 7, 8, 9 and contributed to sections 2, 4, 10, and 11.

Jae Sung Kim - Setup VM on personal server to give access to the entire group for collaborative work. Maintain *nPrint* and pyenv environment on VPN accessed VM. Wrote both pcap_split_1by1.py and pcapsplit.py to be used when developing a *nPrintML* like workflow. Wrote section 2 through 5 of this paper. Researching how to use payload bytes and application packets in *nPrint* to add more features to dataset. Researched methods to mitigate RAM usage to run on lower tier hardware.

References

[1] Pyenv Contributors. *pyenv: Simple Python Version Management*. https://github.com/pyenv/pyenv. Accessed: 2024-05-05.

[2] scikit-learn developers. *GridSearchCV – scikit-learn 1.1.3 documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed: 2024-05-01. 2023.

[3] Jordan Holland et al. "New Directions in Automated Traffic Analysis". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021, pp. 3366–3383. DOI: https://doi.org/10.1145/3460120.3484758.

[4] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*. Canadian Institute for Cybersecurity, University of New Brunswick. Portugal, Jan. 2018.

[5] AutoML Team. *AutoML: Automated Machine Learning*. https://www.automl.org/automl/. Accessed: 2024-05-03. 2024.

[6] Virginia Tech. *Hardware Information for Cluster*. https://wordpress.cs.vt.edu/rlogin/more-information-3/. Accessed: 2024-04-02. 2024.

[7] The nPrint Project. *nPrint*. https://nprint.github.io/nprint/. Accessed: 2024-02-27. 2024.

[8] *Ubuntu 22.04.4 LTS (Jammy Jellyfish)*. https://releases.ubuntu.com/jammy/. Accessed: 2024-04-12. 2024.

[9] XGBoost Development Team. *XGBoost: Scalable Machine Learning System*. https://xgboost.readthedocs.io/en/stable/. Accessed: 2024-05-03. 2024.