KitPlugin Manual

Guy Puts

May 2024

1 Introduction

This manual describes the use of KitPlugin, which provides a testing- and benchmarking-framework for Kitsune IDS, a state-of-the-art anomaly-based Network Intrusion Detection System[1]. For more support, feel free to send an e-mail to guy@guyputs.nl.

2 Getting started

1. Download `https://github.com/GuyPuts/Kitsune-adaptation.git` to a folder on the system.
2. Download and install Python 3.10.
3. Use Visual Studio Installer to download Visual Studio Build Tools 2022. This is necessary to use Cython.
4. Install the required dependencies using pip (requirements.txt is in the root directory of the project).
5. Create the following folders:
    - input_data
    - input_data/attack_types
    - output_data
    - pickles
    - pickles/output_pickles_conv_basis
    - pickles/output_pickles_packet_basis
6. Put the desired .pcap.tsv-files in the input_data folder.
7. Go to example.py and enter the desired commands. Then go to command-line, navigate to the root directory of Kitsune and run `python example.py`.

2.1 Generating features

The first step is to generate features for a certain day of traffic. The easiest way to do this, is by going to example.py, and entering:

```
kitplugin = KitPlugin(input_path=f"input_data/Monday-WorkingHours.pcap.tsv",
    packet_limit=np.Inf, num_autenc=50, FMgrace=None, ADgrace=None,
    learning_rate=0.1, hidden_ratio=0.75)
kitplugin.feature_builder(f"input_data/attack_types/monday_features.csv")
```

Keep in mind that the hyperparameters are not relevant during feature generation. Instead of using a .pcap.tsv-file, it is also possible to use a regular .pcap-file. Kitsune will then convert this into a .pcap.tsv-file. For files with large amounts of packets ($> 5,000,000$), this process will take a **long** time (several days). There are basically two alternatives:

2.1.1 The simple solution

Make use of pre-generated feature-files. Put these files in input_data/attack_types.

2.1.2 The cool but complicated solution

This solution uses *parallelization*. The `netStat`-class contains a function called `updateGetStats`, which computes the damped incremental statistics. This part can be parallelized by commenting-out certain aggregation types. Make sure to also comment-out the variable of the aggregation type (for example HHstat_jit) in the return statement. Run different instances of Kitsune. The simplest way to do this is by opening multiple terminal windows and running `python example.py` multiple times simultaneously. The different feature files can then be patched together **column by column**, for example by using the `replace_entries`-function in example.py. Adapt the function to the number of files that have to be patched together.

2.2 Hyperparameter Optimization

Kitsune will be optimized on different subsets of traffic. Use this block of code in example.py and replace `validation_file`, `UNSW_Benign_small`, and `UNSW_Benign_medium` by the correct files used for validation and training, respectively.:

```
filename = 'input_data/attack_types/validation_file.csv'
data = []
with open(filename, 'r', newline='') as csvfile:
csvreader = csv.reader(csvfile)
for row in csvreader:
    data.append(row)

pickle_file = 'pickles/medium_validate.pkl'
with open(pickle_file, 'wb') as f:
    pickle.dump(data, f)

attacks1 = ["UNSW_Benign_small", "UNSW_Benign_medium"]
for sample in attacks1:
    with open(f"input_data/attack_types/noday_features_{sample}.csv", newline='')
        as csvfile:
        csv_reader = csv.reader(csvfile)
        line_count = sum(1 for row in csv_reader)
print(f'lines: {line_count}')
kitplugin=KitPlugin()
print(f'optimizing kitnet for {sample}')
kitplugin.hyper_opt_KitNET("noday", sample, line_count)
print('done optimizing')
```

The search space can be found on line 742 of KitPlugin.py; change the search space if needed. The study will be written to a .db-file in the root directory of Kitsune. The easiest way to read this is as follows:

```
    # Load Optuna study from the existing database file
    storage = PandasIntegrationStorage("sqlite:///UNSW_Benign_small.db")
    study_name = "UNSW_Benign_small"  # Name of your existing study
    study = optuna.create_study(storage=storage, study_name=study_name, load_if_exists=True)

    # Convert study to a dataframe including user attributes
    df = study.trials_dataframe(include_user_attrs=True)

    # Pickle the dataframe
    with open('optuna_dataframe.pkl', 'wb') as f:
        pickle.dump(df, f)
```

Then, go to the Jupyter-notebook Opt_pickles.ipynb, read the pickles-file, and read the optimal values.

2.3 Training Kitsune

Training Kitsune is done by adapting example.py as follows:

```
    kitplugin = KitPlugin()
    kitplugin.train_kitsune()
```

Adapt `train_kitsune` in `KitPlugin` by adapting the hyperparameters to the correct value.

2.4 Testing Kitsune on attack traffic

After feature files have been generated, Kitsune can be tested on different types of traffic. This is done as follows:

```
    convs = []
    attacks = ["benign - small", "FTP-Patator", "FTP-Patator - Attempted", "SSH-Patator", "SSH-Patat
    for attack in attacks:
        print(attack)
        convs.append(kitTester("tuesday", attack))
```

Viewing the results can be done as follows: Kitsune's root folder contains a Jupyter-notebook called 'Output processing.ipynb'. Copy this file to pickles/output_pickles_conv_basis and open the notebook. This notebook will show the results of running Kitsune on the supplied attack types.

2.5 SHAP-analysis

A SHAP-analysis can be conducted for every day of attack traffic. For example:

```
kitplugin = KitPlugin()
kitplugin.most_significant_packets_sampler("wednesday", 0.0)
results = kitplugin.shap_documenter("wednesday")
```

Keep in mind that for `shap_documenter` to work, the anomaly detector saved in `pickles/anomDetectorFullDataset.pkl` must be the version of Kitsune that we are currently trying to compute SHAP-values for. This file is automatically updated when running `traing_kitsune` in KitPlugin.py. The second parameter of `shap_documenter` is deprecated and thus takes an arbitrary value. The SHAP-results can be analyzed by copying SHAP-analysis.ipynb to pickles/output_pickles_conv_basis and running the cells.

3 Taking subsets of features

To take a subset of features, go to the CSV file shap-reduc.csv. This file contains the list of all features as they are ordered in Kitsune's FE output. Exclude the header and index from zero. Take the indices of the features you want to use, and put them in a Python list. Feed this list into the function `filter_csv_columns` in example.py. The feature mapping process (the function `kitTester`) can now be repeated for the new feature set.

References

[1] Y. Mirsky et al. *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection.* arXiv:1802.09089 [cs]. May 2018. URL: `http://arxiv.org/abs/1802.09089` (visited on 07/12/2023).