SADIT Documentation

Release 1.0.0

Jing Conan Wang

September 23, 2012

CONTENTS

INTRODUCTION

SADIT is a byproduct of research project **A Coordinated Approach to Cyber-Situation Awarness Based on Traffic Anomaly Detection**. It is the acronym of **S**ystematic **A**nomaly **D**etection of **I**nternet **T**raffic. The motivation of SADIT is to make the comparison and the validation of internet anomaly deteciton algorithmes super easy. It provides a collection of Internet Anomaly Detection algorithms proposed by several researchers. Now it contains:

1. Stochastic Anomaly Detector using Large Deviation Theory
2. Deterministic Temporal Anomaly Detector using Support Vector Machine
3. Deterministic Flow by Flow Detector using Support Vector Machine
4. Anomaly Detection Techniques using ART theory

SADIT is not limited to the four algorithms listed above, its utimate goal is to become a standard collection of a variety of internet anomaly detection algorithms.

It also provides an easy-to-use tool to generate labeled flow records, which are helpful in validatation and comparison of methods.

If you are a researcher interested in Internet Anomaly Detection, we strongly encourage you to implement your algorithms following the APIs and data format of SADIT so that you can easily compare your methods with exisiting algorithms in SADIT. Your help will be highly appreciated if you can contribute your own algorithm to the algorithm libray of SADIT. Feel free to contact me if you have any question.

CHAPTER

TWO

WHAT'S NEW

the version 1.0 is a result of big refactor of version 0.0. The refactor makes the code more scalable and less buggy.

- **Paradigm of Object-oriented programming**: The **Configure** module and **Detector** module have been rewritten under object-oriented paradigm. In version 0.0, all modules depends on the global settings file setting.py, which make the code more vulunerable to bugs. In this verison only few scripts depend on settings.py. Classes are widely used to reduce the need to pass parameters around. In case that parameters passing is required, well-defined structures are used.
- **Experiment**: A new folder ROOT/Experiment appears to contain different experiments. You can write your own scripts of Experiment and put them in this folder.
- **Better Sensitivity Anaysis**: In the version 0.0, sensitivity anaysis is done by change the global settings.py file and rerun the simulation. Since settings.py is a typical python module,changing it during the run is really not a good idea. In this version, special Experiment is designed to support sensitivity analysis.

CHAPTER

THREE

STRUCTURE

SADIT consists of two parts. The first part is a collection of anomalies detection algorithms. The second part is labeled flow record generator. The follow sections will describe the two parts accordingly.

3.1 Collection of Anomaly Detection Algorithm

All the detection algorithms locates in the *ROOT/Detector* folder:

- **SVMDetector.py** contains two SVM based anomaly detection algorithmes 1. SVM Temporal Detector and 2. SVM Flow by Flow Detector.
- **StoDetector.py** contains two anomaly detection algorithms based on Large Deviation Theory.

3.2 Labeled Flow Records Generator

Labeled Flow Records Generator consists of a *Configurer* and a *Simulator*. The *Simulator* part is essentially a revised FS Simulator developed by researchers at UW Madison. *Configurer* first generate a flow specification (DOT format) file with certain types of anomalies, then the *Simulator* will generate flow records and corresponding labels.

3.2.1 Configurer

Configurer generate the corresponding DOT file according to description of user behaviour. The important concepts in *Configurer* are as follows:

- **Generator**: description of a certain type of flow traffic. For examples, *Harpoon* generator represents harpoon flows.
- **Behaviour**: description of temporal pattern. There are three types behaviour:
  - **Normal** behaviour is described by start time and duration.
  - **I.I.D behaviour has a list of possible states, but one state** will be selected as current state every *t* seconds according to certain probability distribution.
  - **Markov the state in different time is not independtly and** identically distributed, but is a Markov process
- **Modulator**: combine *Behaviour* and *Generator*, basicially description of generator behaviour. There are three types of modulators, corresponding to three behaviours described above.

3.2.2 Simulator

Simulator is basically a revised version of fs simulator. We have added support to export anoumalous flows(add label information).

CHAPTER

FOUR

USAGE

To run SADIT, just go to the diretory of SADIT source code, change ROOT variable in **settings.py** to the absolute path of the source directory. Then type

```
$ ./run.py
```

in the command line. The help document will come out

usage: run.py [-h] [-e EXPERIMENT] [-i INTERPRETER] [-d DETECT] [-m METHOD] [–data_handler DATA_HANDLER] [–feature_option FEATURE_OPTION] [–export_flows EXPORT_FLOWS] [–entropy_threshold ENTROPY_THRESHOLD] [–pic_name PIC_NAME]

sadit

optional arguments:

-h, --help show this help message and exit

-e EXPERIMENT, --experiment EXPERIMENT specify the experiment name you want to execute. Experiments availiable are: ['Eval', 'Experiment', 'test', 'Sens', 'DetectArgSearch', 'MarkovSens', 'MultiSrvExperiment', 'ImalseSettings', 'MarkovExperiment']. An integrated experiment will run fs-simulator first and use detector to detect the result.

-i INTERPRETER, --interpreter INTERPRETER –specify the interperter you want to use, now support [cpython], and [pypy](only for detector)

-d DETECT, --detect DETECT –detect [filename] will simply detect the flow file, simulator will not run in this case, detector will still use the configuration in the settings.py

-m METHOD, --method METHOD –method [method] will specify the method to use. Avaliable options are: ['svm_fbf': SVMFlowByFlowDetector SVM Flow By Flow Anomaly Detector Method | 'mf': ModelFreeAnoDetector | 'mfmb': FBAnoDetector model free and model based together | 'svm_temp': SVMTemporalDetector SVM Temporal Difference Detector. Proposed by R.L Taylor. Implemented by J. C. Wang <wangjing@bu.ed> | 'mb': ModelBaseAnoDetector]

--data_handler DATA_HANDLER –specify the data handler you want to use, the availiable option are: ['fs_deprec': DataFile from fs output flow file to feature, this class is *depreciated* | 'fs': HardDiskFileHandler Data is stored as Hard Disk File | 'xflow': HardDiskFileHandler_xflow | 'SperottoIPOM2009': SQLDataFileHandler_SperottoIPOM2009 "Data File wrapper for SperottoIPOM2009 format. it is store in mysql server, visit http://traces.simpleweb.org/traces/netfl ow/netflow2/dataset_description.txt for more information | 'pcap2netflow': HardDiskFileHandler_pcap2netflow]

--feature_option FEATURE_OPTION specify the feature option. feature option is a dictionary describing the quantization level for each feature. You need at least specify 'cluster' and 'dist_to_center'. Note that, the value of 'cluster' is the cluster number. The avaliability of other features depend on the data handler.

--export_flows EXPORT_FLOWS specify the file name of exported abnormal flows. Default is not export

--entropy_threshold ENTROPY_THRESHOLD the threshold for entropy,

--pic_name PIC_NAME picture name for the detection result

The help document is quite clear, the only parameter I want clarify is *–experiment*. It specify the experiment you want to execute. An **experiment** is actually a executable python script in Experiment folder. .. I will take .. *Experiment.py* as an example to describle experiment. Avaliable experiments as follows:

- **Experiment.py**: basic experiment which includes configuration, simulation and detection, which corresponds to generating the configuration files, generating the labeled flow records and detectng the anomalies in the records, respectively.
- **Sens.py**: Do sensistive analysis by change the degree of of anomalies, run the detection algorithm accordingly and show results in the same figure.
- **MarkovExperiment.py**: Similar with Experiment, but for Markov type of anomalies
- **MarkovSens.py**: Sensitivity analysis of MarkovExperiment
- **Eval.py**: Evaluation of the detection algorithmm calculate fpr, fnr and plot the ROC curve
- **DetectArgSearch.py**: runs detection algortihms with all combinations of parameters and outputs the results to a folder, helps to select the optimal parameters.

In addition to the parameters in the command line, SADIT has some more tunable parameters in **ROOT/settings.py**. you can customize **SADIT** through changing parameters in **settings.py** file. Since it is a typical python script, so you can use any non-trival python sentence in the **settings.py**.

4.1 Parameters for Labeled Flow Generator

```
NET_DESC = dict(
        topo, # a list of list, the adjacent matrix of the topology
        size, # no. of nodes in the network
        srv_list, # not used
        link_attr_default, # default link attributed, delay, capacity, etc.
        node_type='NNode', # Node type, can be 'NNode', 'MarkovNode', etc.
        node_para, # not used.
        )

NORM_DESC = dict(
        TYPE, # can be 'NORMAL' or 'MARKOV'
        start, # start time, should be string
        node_para, # node parameters for normal case
        profile, DEFAULT_PROFILE, # normal profile
        src_nodes, # node that will send traffic
        dst_nodes, # the destination of the traffic.
        )
```

```
ANO_DESC = dict(
        anoType, # anomaly type, can be anyone defined in  **ano_map** of *Configure/API.py*,
        ano_node_seq, # the sequence no. of the abnormal node
        'T':(2000, 3000), # time range of the anomaly
        .., # anomaly specific  parameters
        )
```

4.2 Parameters for Detectors

```
DETECTOR_DESC = dict(
        interval=20,
        win_size=200,
        win_type='time', # 'time'|'flow'
        fr_win_size=100, # window size for estimation of flow rate
        false_alarm_rate = 0.001,
        unified_nominal_pdf = False, # used in sensitivities analysis
        fea_option = {'dist_to_center':2, 'flow_size':2, 'cluster':3},
        ano_ana_data_file = ANO_ANA_DATA_FILE,
        normal_rg = None, # range for normal traffic, if it none, use the
        whole traffic
        detector_type = 'mfmb', # can be anyone specfied by **detector_map** in *Detector/API.py*
        max_detect_num = None, # the maximum detection number the detector will run
        data_handler = 'fs', # specify the data handler, can be any value defined in
        data_handler_map in Detector/API.py

        #### only for SVM approach #####
        # gamma = 0.01,
        )
```

CHAPTER

FIVE

WANT TO IMPLEMENT YOUR ALGORITHM?

5.1 Use the labeled flow records generator in fs simulator

The generated flows will be the *ROOT/Simulator* folder. The flows end with *_flow.txt*, for example, n0_flow.txt is the network flows trough node 0. File start with *abnormal_* is the exported abnormal flows correspondingly.

A typical line is textexport n0 1348412129.925416 1348412129.925416 1348412130.070733 10.0.7.4:80->10.0.8.5:53701 tcp 0x0 n1 5 4215 FSA

line format prefix nodename time flow_start_time flow_end_time src_ip:src_port->dst_ip:dst_port protocol payload destname unknown flowsize unknown

After finishing your detection algorihms, you need to add the corresponding class name to **detector_map** in *ROOT/Detector/API.py*. Then you can implement your own experiment to compare your algorithms with existing algorithms in SADIT. Look at the sample examples in *ROOT/Experiemnt/* folder. Your can run your experiment by typing

```
./run.py -e <Your Experiment Name>
```

5.2 Use Other flow records

SADIT does not only support the text output format of fs simulator, but also several other types of flow data. The handler classes locate in the *DataHandler.py* and *DataHandler_xflow.py* **DataFile**

- **PreloadHardDistFile** hard disk flow file generated by fs-simulator. It will preload all the flow file into memory, so it cannot deal with flow file larger than your memery
- **PreloadHardDistFile_pcap2netflow** hard disk flow file generated by pcap2netflow tool(the format of flowd-reader)
- **PreloadHardDistFile_xflow**, hard disk flow file generated by xflow tool
- **SQLFile_SperottoIPOM2009**, labeled data stored in mysql server provided by simpleweb.org

The following class is the abstract base class for all data files

```
class Data(object):
    """virtual base class for data. Data class deals with any implementation
    details of the data. it can be a file, a sql data base, and so on, as long
    as it support the pure virtual methods defined here.
    """
    def get_fea_slice(self, rg=None, rg_type=None):
        """ get a slice of feature
        - **rg** is the range for the slice
        - **rg_type** is the type for range, it can be ['flow'|'time']
        """
        abstract_method()

    def get_max(self, fea, rg=None, rg_type=None):
        """ get the max value of feature during a range
        - **fea** is a list of feature name
        - **rg** is the range
        - **rg_type** is the range type
        the output is the a list of element which contains the max
        value of the feature in **fea**
        """
        abstract_method()
    def get_min(self, fea, rg=None, rg_time=None):
        """ get min value of feature within a range. see **get_max** for
        the meaning of the parameters
        """
        abstract_method()
```

It defines the operation we can do with data files. The four algorithmes we implemented requires **get_fea_slice**, **get_max** and **get_min** operations. You can implement more operations, but at least implement these three operations.

Optionally, you can implement a handler class that will manipulate the DataFile and and some useful quantities that may be useful to you algorithms. For example, we implemented **HardDiskFileHandler** with get_em() function to get probability distribution of the flows, which is useful for the stochastic approaches. If you just need the raw data, you can define simple handler class with data file object you want to use as member variable.

Then you just need to add your data_handler to **data_handler_handle_map** defined in *ROOT/Detector/API.py*

CHAPTER

SIX

DOWNLOAD

You can download sadit from here.

or you can user mercurial to get a complete copy with revision history

```
hg clone https://bitbucket.org/hbhzwj/sadit
```

CHAPTER

SEVEN

INSTALLATION

SADIT has been tested on python2.7.2. this software depends on all softwares that fs-simulate depends on:

- ipaddr (2.1.1) Get
- networkx (1.0) Get
- pydot (1.0.2) Get
- pyparsing (1.5.2) Get
- py-radix (0.5) Get

besides: it requires:

- numpy Get
- matplotlib Get
- pygame Get
- profilehooks Get

if you are in debain bases system. you can simple use

```
sudo apt-get install python-dev
sudo apt-get install python-numpy
sudo apt-get install python-matplotlib
sud0 apt-get install python-pygame
```

CHAPTER

EIGHT

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*
- *Glossary*