Code documentation for the anomaly detection module of the VSL

François-Xavier Aubet

1 Introduction

During my bachelor thesis[1] I developed an anomaly detection module for the VSL. It is build to detect anomalies in the connection between the services of the DS2OS. This document is a code documentation of the different parts of this detection system. If you want to learn more about the theoretical background, you can read my bachelor thesis.

The anomaly detection is composed of two different modules: sphinx and echidna. There is one instance on the sphinx on each KA and one instance of Echidna in the whole smart space. Sphinx handles the anomaly detection for each access performed by a service of the DS2OS. Echidna coordinates the Sphinxes.

2 Sphinx

2.1 Workflow

The accesses performed by services of the DS2OS are routed within the VSL by the class "RequestRouter" found in the package "org.ds2os.vsl.ka". Each access is captured and its normality is examined before it is routed, the workflow of this process is presented in figure 1. To do so, the "AccessObserver" class is used as a sort of interface. It contains all the methods used to perform the preprocessing of the accesses. Once the preprocessing is done, the method "logConnection()" is called, it sends the access description to the detection module.

[1] Machine Learning-Based Adaptive Anomaly Detection in Smart Spaces, April 2018

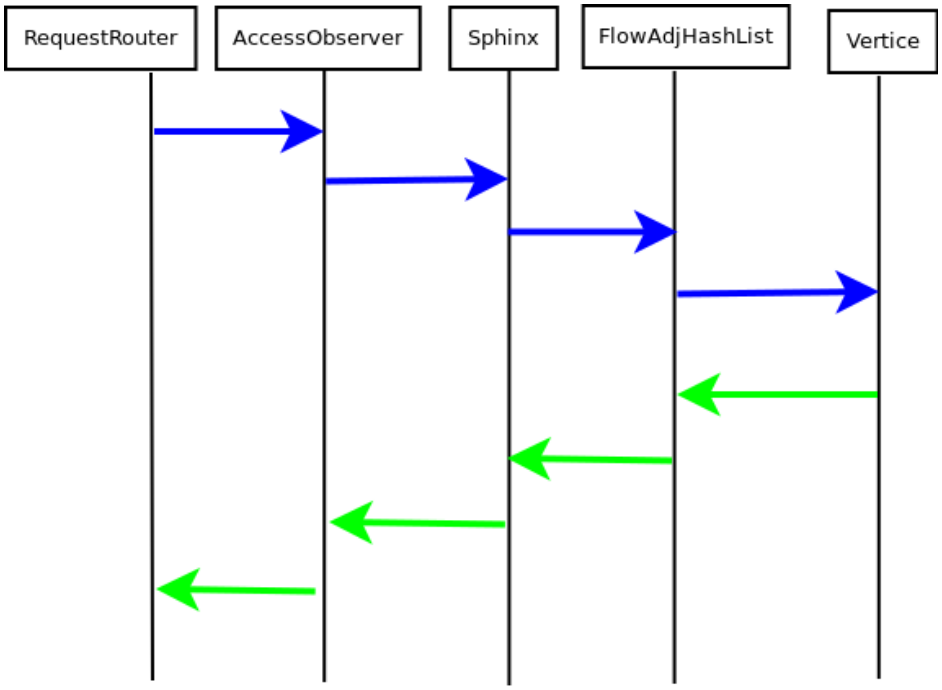

Fig. 1: To access the normality of an access, it is send from the "RequestRouter" through the various classes of the detection module.

The access description is made using an instance of the class "Access" in the package "model".

The first class to receive the access description is the "Sphinx" in package "main". The Sphinx transfers the description to the "FlowAdjHashList" that finds the corresponding sender service. An instance of the class service is created for each service running on the same KA as the Sphinx. In this class, it is assessed if an access should be allowed depending on the known behavior of the service. Then the output of the evaluation is sent all the way back to the "RequestRouter" that can discard the access if the option is set.

2.2 Organization of the Sphinx

In this subsection we review the organization of the classes in the Sphinx.

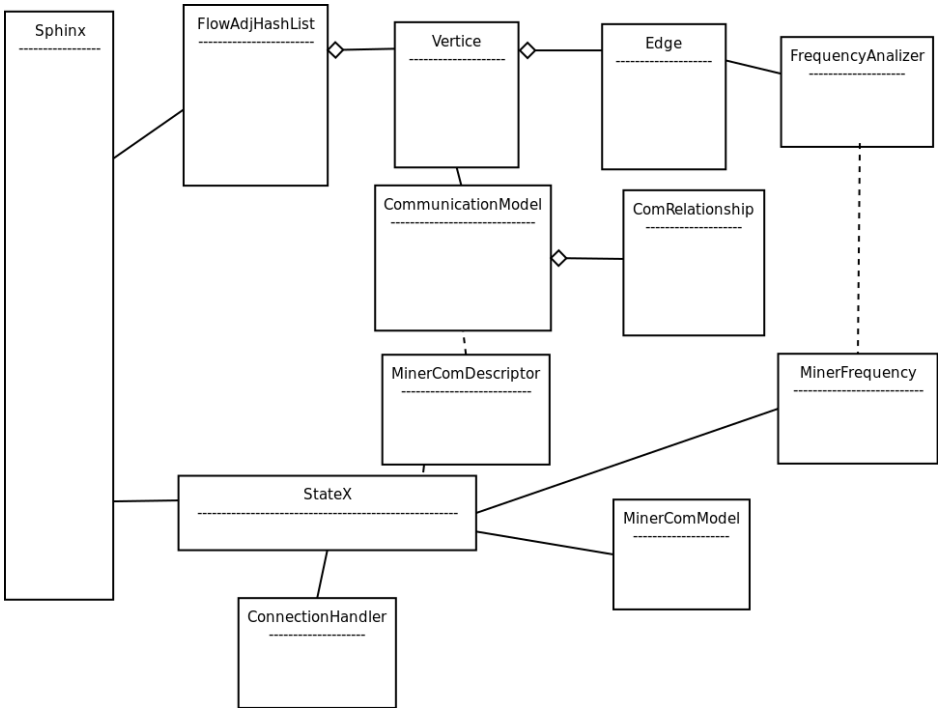

Fig. 2: This is the UML diagram of the main classes of the Sphinx. The upper part of the diagram shows the classes that are used to keep trace of the services' behavior and to assess the accesses. The lower part shows the tasks running in background.

There are two main types of classes in the Sphinx, on one side the classes that are used to save the observations of the services' behavior and to assess the normality of the accesses. These classes are part of the workflow presented in the previous subsection and they are represented in the upper part of the diagram in figure 2. On the other hand, there are classes used to run tasks in the background. They are shown in the lower part of the diagram.

2.2.1 Static classes

The relation between the static classes has been addressed in the workflow. From a structural point of view, the main class, Sphinx, has a FlowAdjHash-List that contains all the vertices of observed by the Sphinx. Each vertex has a list of all the communication relationships that the service has, saved in the form of Edges. Moreover, each vertex has a description of its current behavior contained in the class "CommunicationModel". It is composed of instances of the class "ComRelationship". Each edge has an instance of the "FrequencyAnalizer" class, it is used to mine periodicities in the incoming

accesses.

2.2.2 Miners

The three miners are used to actualize and process information collected by the static classes. They run in the background in order to minimize the computation time needed to evaluate an incoming access. The "MinerFrequency" is used to actualize the periodicity mining done in the "FrequencyAnalizer" class of each edge. The "MinerComDescriptor" goes through all the vertices periodically and actualizes the description of their current behavior contained in the "CommunicationModel" class. Finally, the "MinerComModel" goes through the "CommunicationModel" of the vertices to create an aggregated description corresponding to the typical behavior of each type of service.

2.2.3 Communication

The Sphinx communicates with the Echidna of the smart space via the VSL middleware. There are two parts in this communication. First, the Sphinx publishes the vertices and the edges that it discovers in its VSL nodes. Then, the Sphinx periodically looks for an Echidna service. When it finds it, the sphinx subscribes to the nodes of the Echidna.

3 Echidna

There is one instance of the Echidna service in the whole smart space, it is used to coordinate the Sphinxes. The three main tasks of the Echidna are, to share the knowledge between the Sphinxes, to provide a visualization of what is happening in the smart space and to provide a user interface.

3.0.1 Merging Communication models

The Echidna gathers the communication models from the Sphinxes, merges them and then sends them back. To do so, it periodically looks for Sphinxes in the smart space. When it finds one, it subscribes to the nodes it needs. This way it can gather the communication models of the sphinxes and the description of the edges and vertices. After having merged them, it sends them back by publishing it on his own nodes in the VSL.

3.0.2 Visualization

Echidna offers a visualization of the connection relationships happening in the smart space. This visualization is done using the graphstream library.

Details can be found in the class "GraphVisu".

3.0.3 Antigone

Antigone is an android application used to gather user feedback on the is happening in the smart space. The connection between Echidna and Antigone is done via a TCP socket. Details can be found in the class "SocketManagerServer".

4 Parameters

Two parameters can be set in the configuration file of the KA.

security.sphinx.enabled: if set to one, the sphinx will be started and recieve all the traffic from the KA.

security.sphinx.blockOnAnomaly: if set to one, the accesses detected as anomalous by the sphinx will be blocked before they are routed.

5 Conclusion

These are the most important information. Smaller details are documented in the code with comments. If after reading this and the code you still have some questions please do not hesitate to contact me: francois.aubet@gmail.com.