AIDA Framework

USER MANUAL & DOCUMENTATION

Authors Martin Husák, Jaroslav Kašar, Milan Žiaran

Contact email husakm@ics.muni.cz

Contact address Masaryk University
Institute of Computer Science
Botanická 68a
602 00 Brno

Contents

1 Introduction

AIDA is an analytical framework for processing intrusion detection alerts with a focus on alert correlation and predictive analytics. The framework contains components that filter, aggregate, and correlate the alerts, and predict future security events using the predictive rules distilled from historical records. The components are based on stream processing and use selected features of data mining (namely sequential rule mining) and complex event processing. The framework was designed to be deployed as an analytical component of an alert processing platform. Alternatively, it can be deployed locally for experimentations over datasets.

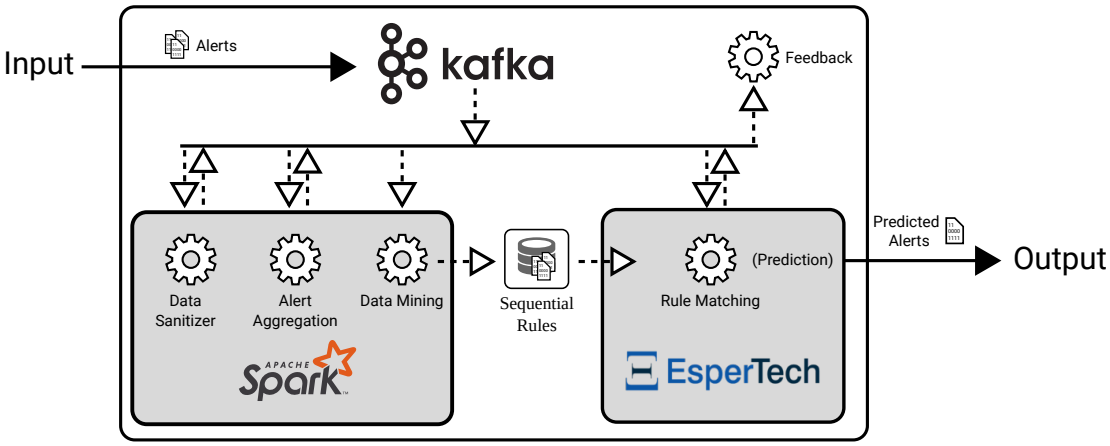


Figure 1: Example of a sequential rule.

Schema of the AIDA framework is presented in Figure 1. The central component of the framework is Kafka message broker[1] that takes the data from inputs and distributes them among the other components while ensuring the correct order of processing of the alerts. Particular components can then be found in two component groups based on their underlying software platform. On the left side, we can see a group of components implemented as applications in Spark analytical engine[2]. These applications typically receive a stream of alerts from Kafka, process them, and return them to Kafka. Some applications, such as the Data Mining component, have another output, such as a database for long-term storage of results of data processing. On the right side of the schema, we can see an application based on Esper[3], another event stream processing framework. The component implemented using Esper uses its features, such as a powerful query language, for the benefits of the whole AIDA framework. The Rule Matching component is also one of the few that provide outputs outside the framework. Outputs are in the same form as inputs, i.e., intrusion detection alerts, so the output channel is implemented analogously. More details on particular components are discussed further in this section.

AIDA framework expects the input alerts to be formatted in IDEA (Intrusion Detection Extensible Alert) format[4]. IDEA is inspired by IDMEF, a popular alert exchange format[5]. Contrary to IDMEF, IDEA is extensible, includes a classification of alerts based

[1] https://kafka.apache.org/
[2] https://spark.apache.org/
[3] http://www.espertech.com/esper/
[4] https://idea.cesnet.cz/en/index
[5] https://www.ietf.org/rfc/rfc4765.txt

on taxonomies from CSIRT communities, and is adjusted to suit practical needs. See the example of an intrusion detection alert in IDEA format in Figure 2.

```
{
  "Format": "IDEA0",
  "ID":"f62537c2-77b8-49c7-a0a2-24c4b81b20f8",
  "CorrelID": [ % IDs of preceding alerts
        "3688762d-2efa-44a8-9ea5-34a57b3ae0c7",
        "ae6d9ac6-6389-407f-9d7e-58b9692c6eaa"
      ],
  "DetectTime": "2019-03-16T12:17:21.609+00:00",
  "Category": ["Attempt.Login"],
  "Confidence": "0.6111",
  "Description": "The source IP address follows a known pattern that is expected to
       continue with the event described in this message.",
  "Note": "OrganizationA.Honeypot1_Recon.Scanning_22, OrganizationB.IDS1_Attempt.
      Login_22 ==> OrganizationA.IDS1_Attempt.Login_22",
  "Source": [{"IP4": ["10.11.12.13"]}],
  "Target": [{"Port":[22]}],
  "Node": [
    { % Node referencing the AIDA Framework
      "Name": "OrganizationX.AIDA",
      "SW": "AIDA",
      "Type": ["Correlation", "Statistical"]
    },
    { % Node derived from the rule
      "Name": "OrganizationA.IDS1"
    }
  ]
}
```

Figure 2: Example of an intrusion detection alert in IDEA format.

AIDA framework extracts sequential rules from the intrusion detection alerts. The sequential rules are then used to predict upcoming attacks. An example of a sequential rule is presented in Figure 3. It consists of ordered n-tuples (sensor name, type of event, destination port) and support and confidence values. The n-tuples fold two groups; the first group implies the second group. Support value indicates the number of sequences in the sequential database that conform to the rule divided by the total number of all the sequences. The confidence value is a conditional probability of the second part of the rule given the first part of the rule.

```
OrganizationA.Honeypot1_Recon.Scanning_22,
OrganizationB.IDS1_Attempt.Login_22
==>
OrganizationA.IDS1_Attempt.Login_22
#SUPP: 0.0011 #CONF: 0.6111
```

Figure 3: Example of a sequential rule.

The remainder of this documents contains a user guide, installation instructions, and technical documentation. See the user guide for a quick start on how to work with AIDA framework, either via command line or via graphical interface. The quick start should be sufficient to those users, who wish to run AIDA framework locally, e.g., to experiment over datasets. Installation instructions are recommended for users who want to deploy AIDA framework in their environment, and run it operationally. Finally, the technical documentation is intended for developers and advanced users interested in the implementation and exact procedures executed by the framework.

2 User Guide

The user has two options on how to use the AIDA framework. In he first part of the user guide, we provide a quick start using a command line that provides full control over the data and their processing. The second part of user guide descirbes quick start using web interface (dashboard) that is recommended for users that need only basic functionality or prefer data visualizations.

2.1 Quick Start Using Command Line

For a quick start using command line, go to the provision directory and run Vagrant:

```
cd provision
vagrant up
```

AIDA framework will start in few minutes. Then, send your data to the framework using the following command (you need to have netcat installed):

```
nc localhost 4164 < path_to_file_with_your_data
```

If you do not have your own data, we recommend trying AIDA framework out with our dataset[6]. Download and unzip the main file in the datase (dataset.idea.zip) and use it in the command above.

Run data mining

Trigger the data mining procedure (otherwise, it starts every 24 hours that you would have to wait):

```
sudo systemctl start mining
```

Check the logs of the data mining component:

```
sudo journalctl -u mining
```

Update rules

Open the database with the mined rules:

```
sqlite3 /var/aida/rules/rule.db
```

Check the rules in the database:

```
select * from rule;
```

Activate all the rules so that they are used by the rule matching component:

```
update rule set active=1;
```

Restart matching component to start matching activated rules:

```
sudo systemctl restart matching
```

[6] http://dx.doi.org/10.17632/p6tym3fghz.1

Send some more data into AIDA, they will be matched against the rules to predict upcoming events:

```
nc localhost 4164 < path_to_file_with_your_data
```

Check outputs

Predicted rules are saved in the root directory of this repository in ‘predictions.json‘ file. You can also get the predictions directly from Kafka:

```
‘/opt/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
    predictions --from-beginning‘
```

2.2 Quick Start Using AIDA Dashboard

For a quick start using AIDA dashboard, go to the provision directory and run Vagrant:

```
cd provision
vagrant up
```

AIDA framework will start in few minutes. When everything is up and running, open the dashboard on the following URL:

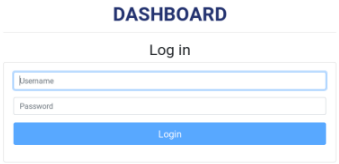
```
http://localhost:8080/
```

You should see the login page. Type in the following default credentials:

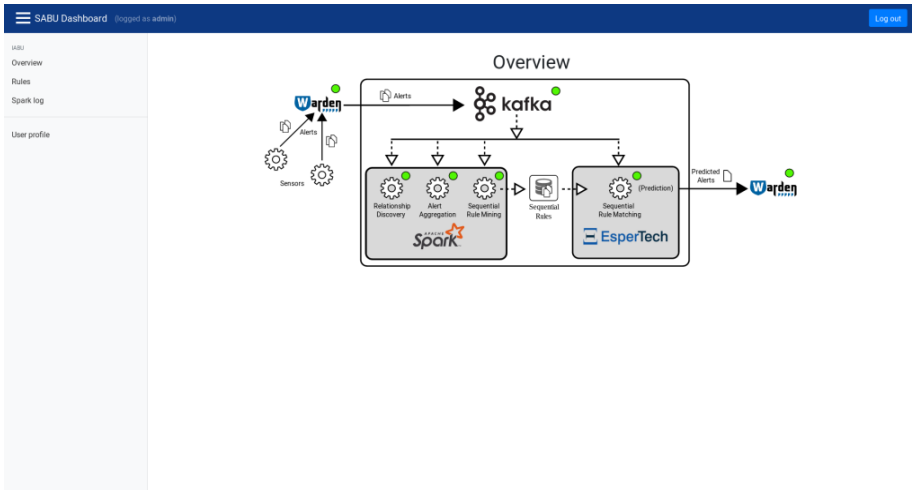
Username: admin

Password: aida.admin

The login page looks as follows:

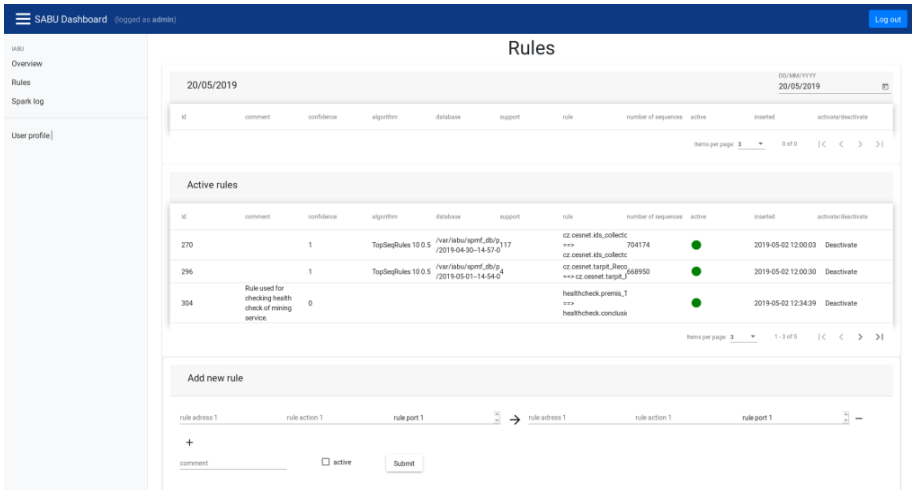


After the login, you should the Overview panel with the status of the components of the system. Green circles indicate running component, red indicated problems with the component. Hover cursor over the icon of the component, a hover text with status text and last lines in log of the component will appear. If you see black circles, there are no information on status available. In such case, check the system.

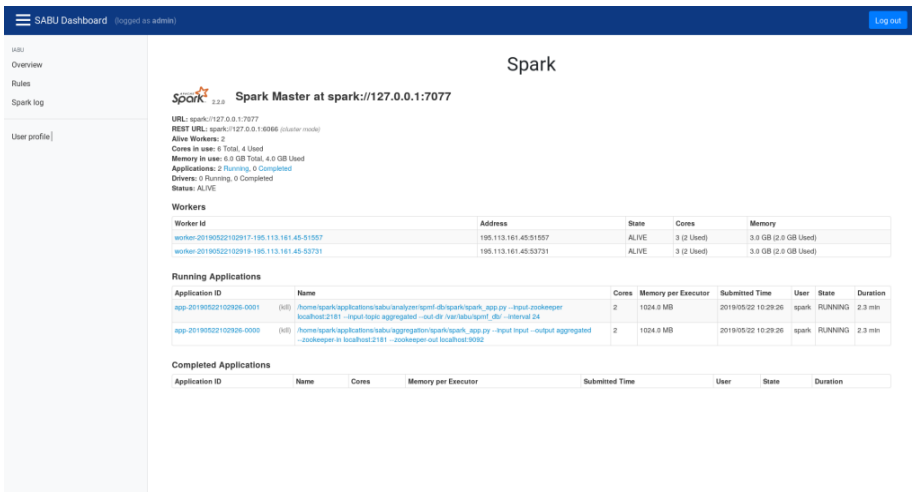


On the left side of the screen, you can see a navigation panel with links to other tabs. The Rules tab allows the user to go through the mined rules in and activate or inactivate

them. The first table from top of the poge shows the rules mined on a given day. Default is the current day, user may change the day usign the calender in the right top corner. The second table shows all the active rules. Both tables show read or green circles that indicate if the rule is active or not. Click them to change the status and activate or inactivate the rule on the line. Finally, at the bottom of the page, there is a box where users may enter their own rules. Fill all the fields and click Submit to add your custom rule to the database.



Users interested in performance of the framework should see the panel Spark log, where they can access the user interface of Spark framework, on which many components are based. Information on the umber and load of workers, memory consumptions, and time to finish jobs are presented there.



Finally, for the configuration of the dashboard, visit the following URL:

```
http://localhost:8000/admin/
```

to access the configuration page. Use the same credentials as for the dashboard, the defaults are:

Username: admin

Password: aida.admin

In the ocnfiguration, you can add or remove user accounts, and customize the framework.

3 Installation Instructions

Simplest way how to install and use AIDA is by running vagrant box `vagrant up` from `provision` directory. If you want to install AIDA on an already existing machine, you can follow the following instructions. There is a Quick Installation section whereby you can install AIDA just by running Ansible playbook. Or you can install AIDA manually step by step from the instructions in Manual Installation section.

3.1 Prerequisites

Installed Ubuntu 18.04.2 and AIDA repository downloaded or cloned into `/vagrant` directory. You can choose any other directory, but we will refer the repository root as `/vagrant`. You should have also installed the following dependencies:

- Installed Java JDK version 1.8
- Installed Maven version 3.6
- Installed Python 2.7 as `python`
- Installed Python 3.6 as `python3`
- Installed Pip for python 2.7 as `pip`
- Installed Pip for python 3.6 as `pip3`

You can install the above prerequisites by running:

```
$ sudo apt-get update
$ sudo apt-get -y install openjdk-8-jdk maven python-pip python3-pip
$ sudo update-java-alternatives --set java-1.8.0-openjdk-amd64
```

3.2 Quick Installation

The simplest way how to install AIDA is by running Ansible playbook.
You have to have Ansible installed. You can do it by running following command:

```
$ sudo apt-get -y install ansible
```

Then run ansible playbook:

```
$ cd /vagrant/provision/ansible
$ ansible-playbook -i inventory-local.ini install-aida.yml
```

Note that the Ansible will also install Java 8, Python 3 and other packages listed in prerequisites section.
You can check that the main AIDA services are up and running by running.

```
$ sudo systemctl status aida-input sanitization aggregation matching
```

3.3 Manual Installation

This section provides step-by-step instructions on how to install AIDA framework on an existing machine without ansible. This process is more complicated and, thus, is recommended only to advanced users, and users interesting in implementation of the framework. During the installation, we are going to create dedicated user accounts for almost every AIDA service. However, you can create just one user for all the AIDA services, and run the framework under on user.

3.3.1 Install Kafka

We will install Kafka and Zookeper and run single Kafka broker on the machine.
First, create Kafka user:

```
$ sudo useradd -M kafka
```

Install Zookeper service:

```
$ sudo apt-get -y install zookeeperd
```

Download Kafka binary into `/opt/kafka`:

```
$ sudo su
# mkdir /opt/kafka
# cd /opt/kafka
# wget https://archive.apache.org/dist/kafka/2.3.0/kafka_2.12-2.3.0.tgz -O kafka.
    tgz
# tar -xzf kafka.tgz --strip-components=1
# rm kafka.tgz
# chown -R kafka:kafka ./
```

Create systemd unit file for `kafka` service:

```
$ sudo cp /vagrant/provision/ansible/roles/kafka/templates/kafka.service.j2 /etc/
    systemd/system/kafka.service
```

Then you have to replace following placeholders in the newly created file `/etc/systemd/system/kafka.service` as follows:

- `{{ kafka_user }}` with `kafka` or other user you had chosen to use
- `{{ kafka_dir }}` with `/opt/kafka`

Create systemd unit file for `kafka-topic` services:

```
$ sudo cp /vagrant/provision/ansible/roles/kafka/templates/kafka-topic.service.j2 /
    etc/systemd/system/kafka-topic@.service
```

Then, replace the same placeholders as in previous step in the newly created file.
Now you can start the `kafka` service by running:

```
$ sudo systemctl start kafka
```

And check that it's running:

```
$ sudo systemctl status kafka
```

Also check that `kafka-topic` service is working by creating `input` topic:

```
$ sudo systemctl start kafka-topic@input
$ sudo systemctl status kafka-topic@input
```

3.3.2 Install Spark

We will install and configure Spark so we are able to run Spark applications in local mode.
First, create Spark user:

```
$ sudo useradd -M spark
```

Download Spark binary:

```
$ sudo su
# mkdir -p /opt/spark/spark
# cd /opt/spark/spark
# wget https://archive.apache.org/dist/spark/spark-2.4.3/spark-2.4.3-bin-hadoop2.7.
    tgz -O spark.tgz
# tar -xzf spark.tgz --strip-components=1
# rm spark.tgz
# wget https://search.maven.org/classic/remotecontent?filepath=org/apache/spark/
    spark-streaming-kafka-0-8-assembly_2.11/2.4.3/spark-streaming-kafka-0-8-
    assembly_2.11-2.4.3.jar -O /opt/spark/spark/jars/spark-streaming-kafka-0-8-
    assembly_2.11-2.4.3.jar
```

Prepare modules for Spark applications:

```
$ sudo su
# mkdir -p /opt/spark/applications
# cp -r /vagrant/commons/commons-python /opt/spark/applications/modules
# pip3 install -r /opt/spark/applications/modules/requirements.txt
# apt-get -y install zip
# zip -j -r /opt/spark/applications{.zip,}
```

Prepare script for running Spark applications:

```
$ sudo su
# cp /vagrant/provision/ansible/roles/spark/templates/run-application.sh.j2 /opt/
    spark/applications/run-application.sh
# chmod +x /opt/spark/applications/run-application.sh
```

Now, you have to replace placeholder `{{ spark_home_dir }}` with `/opt/spark/spark` in the `run-application.sh` file.
Create configuration file:

```
$ sudo cp /vagrant/provision/ansible/roles/spark/templates/spark-defaults.conf.j2 /
    opt/spark/spark/conf/spark-defaults.conf
```

Replace placeholder `{{ spark_modules_zip }}` with `/opt/spark/applications/modules/modules.zip` in the newly created configuration file. You can also increase the `spark.driver.memory` due to available memory to be able to pass more data into AIDA.
Make `spark` user owner of the `/opt/spark` directory:

```
$ sudo chown -R spark:spark /opt/spark/
```

3.3.3 Install Java commons for AIDA components

Install Java commons modules for AIDA components:

```
$ cd /vagrant/commons/commons-idea
$ mvn clean install
$ cd /vagrant/commons/commons-mining
$ mvn clean install
```

3.3.4 Install aida-input service

Install `aida-input` service which will listen on port 4164 and will send all the incoming events into AIDA:

```
$ sudo cp /vagrant/provision/ansible/roles/aida-input/templates/aida-input.service.
    j2 /etc/systemd/system/aida-input.service
```

Replace following placeholders in the newly created file `/etc/systemd/system/aida-input.service` as follows:

- `{{ kafka_user }}` with `kafka` or other user you had chosen to use while installing Kafka
- `{{ kafka_dir }}` with `/opt/kafka`
- `{{ aida_input_port }}` with `4164`

Run the `aida-input` service:

```
$ sudo systemctl start aida-input
```

Check that the service is running:

```
$ sudo systemctl status aida-input
```

3.3.5 Install Sanitization component

Create sanitization user:

```
$ sudo useradd -M sanitization
```

Install sanitization:

```
$ sudo mkdir /opt/sanitization
$ cd /vagrant/services/sanitization
$ mvn clean package
$ cp ./target/*jar-with-dependencies.jar /opt/sanitization/sanitization.jar
$ sudo chown -R sanitization:sanitization /opt/sanitization
```

Create systemd unit file for `sanitization` service:

```
$ sudo cp /vagrant/provision/ansible/roles/sanitization/templates/sanitization.
    service.j2 /etc/systemd/system/sanitization.service
```

Then, replace the following placeholders in the newly created file:

- `{{ sanitization_user }}` with `sanitization` or other user you had chosen
- `{{ sanitization_dir }}` with `/opt/sanitization`

Run the `sanitization` service:

```
$ sudo systemctl start sanitization
```

Check that the service is running:

```
$ sudo systemctl status sanitization
```

3.3.6 Install Aggregation component

Put Aggregation Spark application into the required directory:

```
$ sudo cp -r /vagrant/services/aggregation /opt/spark/applications/aggregation
$ sudo chown -R spark:spark /opt/spark/applications/aggregation
```

Install python dependencies:

```
$ sudo pip3 install -r /opt/spark/applications/aggregation/requirements.txt
```

Create systemd unit file for `aggregation` service:

```
$ sudo cp /vagrant/provision/ansible/roles/aggregation/templates/aggregation.
    service.j2 /etc/systemd/system/aggregation.service
```

Then replace following placeholders in newly created file:

- `{{ spark_user }}` with `spark`
- `{{ spark_run_app_script }}` with `/opt/spark/applications/run-application.sh`
- `{{ aggregation_install_dir }}` with `/opt/spark/applications/aggregation`

Run the `aggregation` service:

```
$ sudo systemctl start aggregation
```

Check that the service is running:

```
$ sudo systemctl status aggregation
```

3.3.7 Prepare Rules database

We are going to create SQLite3 database file with `rule` table for mined rules.
First, create the directory:

```
$ sudo mkdir -p /var/aida/rules
```

Install SQLite3 and create database file with required permissions:

```
$ sudo apt-get -y install sqlite3
$ sudo sqlite3 /var/aida/rules/rule.db 'CREATE TABLE rule (id INTEGER PRIMARY KEY,
    inserted DATETIME DEFAULT CURRENT_TIMESTAMP, rule TEXT NOT NULL, support
    INTEGER, number_of_sequences INTEGER, confidence REAL, active INTEGER DEFAULT
    0, comment TEXT, database TEXT, algorithm TEXT )'
$ sudo chmod 0666 /var/aida/rules/rule.db
```

3.3.8 Install mining service

Create mining user:

```
$ sudo useradd -M mining
```

Install mining binary:

```
$ sudo mkdir /opt/mining
$ cd /vagrant/services/mining
$ mvn clean
$ mvn package
$ sudo cp ./target/*jar-with-dependencies.jar /opt/mining/mining.jar
$ sudo chown mining:mining /opt/mining
```

Create systemd unit file for `mining` service:

```
$ sudo cp /vagrant/provision/ansible/roles/mining/templates/mining.service.j2 /etc/
    systemd/system/mining.service
```

Then replace following placeholders in newly created unit file:

- `{{ mining_user }}` with `mining` or other user you had chosen
- `{{ mining_dir }}` with `/opt/mining`
- `{{ database_path }}` with `/var/aida/rules/rule.db`

Create systemd timer for `mining` service:

```
$ sudo cp /vagrant/provision/ansible/roles/mining/templates/mining.timer.j2 /etc/
    systemd/system/mining.timer
```

Note that mining is a one-shot service. It will run the mining on all event currently available in Kafka and then finish. You can trigger the mining by starting the service:

```
$ sudo systemctl start mining
```

Then, check it did exit successfully:

```
$ sudo systemctl status mining
```

Alternatively, you can check the logs:

```
$ sudo journalctl -u mining
```

3.3.9 Install Matching component

Create user for the Matching component:

```
$ sudo useradd -M matching
```

Install binary for the Matching component:

```
$ sudo mkdir /opt/matching
$ cd /vagrant/services/matching
$ mvn clean package
$ sudo cp ./target/*jar-with-dependencies.jar /opt/matching/matching.jar
$ sudo chown -R matching:matching /opt/matching
```

Create systemd unit file for `matching` service:

```
$ sudo cp /vagrant/provision/ansible/roles/matching/templates/matching.service.j2 /
    etc/systemd/system/matching.service
```

Then replace following placeholders in newly created file:

- `{{ matching_user }}` with `matching` or other user you had chosen
- `{{ matching_dir }}` with `/opt/matching`
- `{{ database_path }}` with `/var/aida/rules/rule.db`

Run the `matching` service:

```
$ sudo systemctl start matching
```

Check that the service is running:

```
$ sudo systemctl status matching
```

3.3.10 Install aida-output service

The `aida-output` service will read predictions from Kafka and save them into file. First, create predictions file with permission of same user used for Kafka:

```
$ touch /vagrant/predictions.json
$ chown kafka:kafka /vagrant/predictions.json
```

Create systemd unit file for `aida-output` service:

```
$ sudo cp /vagrant/provision/ansible/roles/aida-output/templates/aida-output.
    service.j2 /etc/systemd/system/aida-output.service
```

Replace following placeholders in the newly created unit file:

- `{{ kafka_user }}` with `kafka` or other user you had chosen to use while installing Kafka
- `{{ kafka_dir }}` with `/opt/kafka`
- `{{ predictions_file_path }}` with `/vagrant/predictions.json`

Run the `aida-output` service:

```
$ sudo systemctl start aida-output
```

Check that the service is running:

```
$ sudo systemctl status aida-output
```

3.3.11 Install REST API

REST API is used for communications with the AIDA Dashboard. The API is open on port 7777. This service is not mandatory, if you are not planning to use Dashboard you can skip this part.
REST API is implemented as a Django application which will run on Apache2 with WSGI. First, install Apache2 with WSGI support for Python 3:

```
$ sudo apt-get -y install apache2 libapache2-mod-wsgi-py3
```

REST API is getting information about the AIDA services via dbus systemd interface, so we will have to install dbus dependencies:

```
$ sudo apt-get -y install dbus libdbus-glib-1-dev libdbus-1-dev python-dbus
```

Now, place the Django application into correct directory:

```
$ sudo cp -r /vagrant/services/restapi  /var/www/restapi
```

Create python virtual environment for REST API:

```
$ sudo su
# apt-get -y install python3-venv
# cd /var/www/restapi
# sudo python3 -m venv venv --system-site-packages
# source venv/bin/activate
# pip install -r requirements.txt
# deactivate
```

Create config file for the application:

```
$ sudo mkdir /etc/aida/
$ sudo cp /vagrant/provision/ansible/roles/restapi/templates/rest.ini.j2 /etc/aida/
    rest.ini
```

and replace the placeholder `{{ database_path }}` in the file with `/var/aida/rules/rule.db`.
Now, create the config file for Apache:

```
$ sudo cp /vagrant/provision/ansible/roles/restapi/templates/apache2.conf.j2 /etc/
    apache2/sites-enabled/rest-aida.conf
```

Replace placeholders in newly created config file

- `{{ restapi_dir }}` with `/var/www/restapi`
- `{{ restapiaida_dir }}` with `/var/www/restapi/restapiaida`
- `{{ venv_dir }}` with `/var/www/restapi/venv`

Finally, restart Apache2:

```
$ sudo systemctl restart apache2
```

and check that REST API is responding on address `localhost:7777`. For example, the following command:

```
$ curl http://localhost:7777/componentsinfo
```

should return a JSON with status of the AIDA components.

4 Technical Documentation

This section contains the technical documentation for the AIDA framework. The documentation is provided separately for each component, starting with the description of the inputs and outputs.

4.1 Inputs and Outputs

Simplest way for the alerts to enter AIDA framework is via TCP socket on port 4164. AIDA contains systemd service `aida-input`, which is simple unix pipeline of Netcat listening on TCP socket and forwarding the data into Kafka via Kafka console producer. This alerts are being send into Kafka topic `input`. There are expected one IDEA alert per line. You can of course skip this service and send the data directly into the topic by yourself.
The predicted alerts are stored in file `predictions.json` in the root directory of repository. There is systemd service called `aida-output` which is also simple unix pipeline reading alerts from Kafka topic `predictions` and then saving them into the file. You can of course skip this service and consume the data directly from the Kafka topic.

4.2 Kafka

Kafka is used as the primary communication channel between components and the way of moving alerts through the system. Kafka is installed in version 2.3.0 in the directory `/opt/kafka/`. Managing of Kafka broker is done via systemd service `kafka`. For managing of Kafka topics is created systemd template service `kafka-topic@`.
AIDA uses the following Kafka topics for asynchronous communication:

- input,
- sanitized,
- aggregated,
- predictions,
- observations.

4.3 Spark

Spark is installed in version 2.4.3 with extension for Kafka streaming. Because AIDA is using just one machine, spark applications are being executed in local mode.
It the case a Dashboard will run on different machine, you will need to set CORS headers into Spark UI. This can be done by implementing simple Java Servlet Filter. Put jar with the filter into `/opt/spark/spark/jars` directory. Then set the filter by specifying the filter class name in `spark.ui.filters` configuration in the `/opt/spark/spark/conf/spark-defaults.conf` file.

4.4 Sanitizer

The Sanitizer is a stream-processing tool for syntactic and semantic checks of the alerts on the input. Firstly, Sanitizer drops alerts which do not comply with a IDEA format. Secondly, it runs several semantics checks and drops or modify alerts of no interest in AIDA. Lastly, it remove unnecessary IDEA fields, which are not being used in AIDA, to to speed up data processing in the whole framework.

Following semantic checks are being applied:

- Drop alerts with category `Vulnerable` or `Abusive.Sexual`.
- Drop alerts with missing source IPv4.
- Remove nodes without name or with name containing `warden_filer`.
- Remove `Test` category from the alerts.

The Sanitizer is implemented as Kafka Streaming application which reads alerts from `input` topic and writes processed alerts into `sanitized` topic. Managing of this component is done via systemd service `sanitization`.

4.5 Aggregation

Task of Aggregation component is to find and mark redundant alerts so that these are not included in further analyses. AIDA is primarily focused on the aggregation of following two types of aggregates, duplicates and continuing alerts. Aggregation is a Spark Streaming application which runs in local mode (there is no necessity for running the whole Spark cluster with slaves) and is managed by the systemd service called `aggregation`.

4.5.1 Duplicates

Duplicates are processed in the batches with size of one minute, windows size of five minutes and sliding size of one minute. Two alerts are considered to be duplicates if their `source IPv4`, `destination IPv4`, `alert category`, `detecting node name` and `detect time` equals and both appear in the five minute window. In this case one of these alerts is considered to be duplicate and AIDA duplicate field in the alert `$._aida.Duplicate` is being set to `true`.

4.5.2 Continuing alerts

Continuing alerts are processed in the batches with size of one minute, windows size of seventy minutes and sliding size of one minute. The alert is considered to be continuing when it has the same `source IPv4`, `destination IPv4`, `destination port`, `alert category`, `detecting node name` as some previous alert in the seventy minutes window. In this case the alert is considered to be a continuing and the value of AIDA continuing field in the alert `$._aida.Continuing` is being set to the ID of the first alert with the same properties in the window.

4.6 Data Mining

Data Mining component generate sequential rules and stores them into database for late use by Rule Matching component.
The component read all alerts currently buffered in `aggregated` Kafka topic. Retention of the topics is set to one day by default, so the Data Mining component is working with alerts from the last day. The component builds the sequential database from these alerts and then run algorithms on these sequences to generate rules.
A sequence is a vector of items (n-tuples) that share a common property. In our implementation we chose pair of source IPv4 and target IPv4 as a common property, so the sequences represents behaviour of attacker in regards to a specific target. The item consists of an alert type, target port, and name of the node.

The rules are generated by using the SPMF library. We are currently using Top Sequential Rules algorithm with `K` being set to 10 and minimal confidence being set to 0.5.
Mined rules are stored into sqlite database `/var/aida/rules/rule.db`, which is in more detail described in the next section.
The component can be run by oneshot systemd service called `mining`. Systemd timer is created for the purposes of running Mining automatically in defined interval. The timer interval is being set to one day, which corresponds to the Kafka retention interval.

4.7 Database

We are using sqlite3 database for storing and managing rules. It can be found in the `/var/aida/rules/rule.db` file. The database contains main table called `rule` described in Table 1. Data Mining component adds new rules into the table or rules can be add manually. Then it's on the user of AIDA to activate rules he is interested in. Activated rules are going to be used in Rule Matching component to predict and aggregated alerts.

Table 1: Database scheme for storing sequential rules.

Item	Type	Example
id	INT	123
date	CHARACTER (10)	"2018-10-20"W
rule	VARCHAR (255)	"OrganizationA.Honeypot1_Recon.Scanning_22, OrganizationB.IDS1_Attempt.Login_22 => OrganizationA.IDS1_Attempt.Login_22"
support_hits	INT	42
support_total	INT	100,000
confidence	REAL	0.6789
active	INT	0
comment	VARCHAR (255)	"This is a sample rule, do not use."

4.8 Rule Matching

Rule Matching component processes the stream of alters and queries them for the presence of predefined rules. If the predefined rules is matched, an action is performed.
The Rule Matching component is written as Esper streaming application. On the startup, the component reads active rules from the database and generate two EPL queries for each active rule. These queries are then deployed into the Esper runtime engine. Listener, creating predicted alerts and sending them to Kafka, is attached to the queries. All the queries are operating on one hour long sliding window Figure 4, which also filters alerts of no interest for Matching, like duplicates and continuing aggregates.
There are two EPL queries created for each active rule.
The first query is marked as `PREDICTION` and is matching the first part of the rule Figure 5. When the first part is matched, the attached listener is invoked. The listener creates predicted alerts, based on the alerts that appear in the second part of the rule, and sends them into Kafka topic `predictions`.
The second query is marked as `OBSERVATION`. It matches the whole rule. When the entire rule is matched, the observation listener is invoked. The listener sends alert into Kafka

```
@public create window IdeaWindow.ext:time_order(detectTime.getTime(),1 hours) as
    select * from Idea
insert into IdeaWindow
select * from Idea
where
source is not null and
(source.where(x => x.IP4 is not null).anyOf(x => (x.IP4.countOf())>0)) and
target is not null and
(target.where(x => x.IP4 is not null).anyOf(x => (x.IP4.countOf())>0)) and
(additionalProperties is null or (additionalProperties.get("_AIDA")) is null or
(cast(additionalProperties.get("_AIDA"),java.util.Map).get("Duplicate")) is null
    and
(cast(additionalProperties.get("_AIDA"),java.util.Map).get("Continuing")) is null)
```

Figure 4: Filling the time window for optimized pattern matching in EPL.

topic called `observations`, noticing that the predicted alert from the previous query was confirmed.

```
@Rule(
value='OrganizationA.Honeypot1_Recon.Scanning_22, OrganizationB.IDS1_Attempt.
    Login_22 ==> OrganizationA.IDS1_Attempt.Login_22',
confidence=0.6111,
type=StatementType.PREDICTION)
select * from IdeaWindow(
(category.contains("Recon.Scanning")) and (node.anyOf(x => x.name="OrganizationA.
    Honeypot1")) and (target.anyOf(x => x.port.contains(22))) or
(category.contains("Attempt.Login")) and (node.anyOf(x => x.name="OrganizationB.
    IDS1")) and (target.anyOf(x => x.port.contains(22)))
)
match_recognize (
partition by source[0].IP4[0], target[0].IP4[0]
measures A[0] as A, B[0] as B
pattern (match_recognize_permute(A+,B+))
define
A as (A.category.contains("Recon.Scanning")) and (A.node.anyOf(x => x.name="
    OrganizationA.Honeypot1")) and (A.target.anyOf(x => x.port.contains(22))),
B as (B.category.contains("Attempt.Login")) and (B.node.anyOf(x => x.name="
    OrganizationB.IDS1")) and (B.target.anyOf(x => x.port.contains(22)))
)
```

Figure 5: Predictive rule converted into optimized EPL query.

Rule Matching component is managed by the systemd service called `matching`.

4.9 REST API

REST API is implemented using Django framework and offers several endpoint that allow communication and data exchange between dashboard and the rest of the framework. The following endpoints are implemented in the REST API:

- GET /componentsinfo/ – extended status of AIDA components, including indicators of application running, systemd status, and last 5 lines of logs.
- GET /db/rules/YYYY-MM-DD/ – returns the list of rules mined on the given day.
- GET /db/activerules/ – returns the list of active rules, i.e., rules with `active` flag set.
- POST /db/activerules/ID/ – activates the rule with given ID.
- POST /db/inactiverules/ID/ – deactivates the rule with given ID.
- POST /db/deleterules/ID/ – deletes rule with given ID.

- POST /db/addrules/pravidlo/ – creates new rule.
- /enforcedatamining/ – enforces the data mining procedure over the current sequential database and formats it.
- /reloadrulematching/ - enforces restart of Rule Matching component that loads set of currently active rules.

5 Acknowledgment

The authors of the AIDA framework are Martin Husák, Jaroslav Kašar, and Milan Žiaran. The authors would like to thank Michal Pavúk, Vít Rusňák, and Jakub Kolman for the dashboard, and Petr Velan and Samuel Šuľan for testing the framework.

The design of the AIDA framework is described in the research paper:

```
Martin Husák and Jaroslav Kašpar. 2019. AIDA Framework: Real-Time Correlation and
    Prediction of Intrusion Detection Alerts. In Proceedings of the 14th
    International Conference on Availability, Reliability and Security (ARES '19).
    ACM, New York, NY, USA, Article 81, 8 pages. DOI: https://doi.org
    /10.1145/3339252.3340513
```

To cite the paper, we recommend using the following bibtex entry:

```
@inproceedings{AIDAframework,
 author = {Hus\'{a}k, Martin and Ka\v{s}par, Jaroslav},
 title = {AIDA Framework: Real-Time Correlation and Prediction of Intrusion
     Detection Alerts},
 booktitle = {Proceedings of the 14th International Conference on Availability,
     Reliability and Security},
 series = {ARES '19},
 year = {2019},
 isbn = {978-1-4503-7164-3},
 location = {Canterbury, CA, United Kingdom},
 pages = {81:1--81:8},
 doi = {10.1145/3339252.3340513},
 publisher = {ACM},
 address = {New York, NY, USA}
}
```

The development of the framework and related research were supported by the Security Research Programme of the Czech Republic 2015 - 2020 (BV III / 1 VS) granted by the Ministry of the Interior of the Czech Republic under No. VI20162019029 The Sharing and analysis of security events in the Czech Republic.