Monash University

FIT5202 - Data processing for Big Data

Assignment 2: Detecting Linux system hacking activities Part B

Due: Sunday, Nov 1, 2020, 11:55 PM (Local Campus Time)
Worth: 10% of the final marks

Background

StopHacking is a start-up incubated in Monash University to develop cloud service to detect and stop computer hackers. Although they have some rule-based service to identify certain hacks, they would like to add machine learning models which can integrate with their Spark cluster to process large amounts of data and detect any potential hacks. They hired us as *the Analytics Engineer* to investigate the open data from the Cyber Range Labs of UNSW Canberra and build models based on the data to identify abnormal system behaviour. In addition, they want us to help them integrate the machine learning models into the streaming platform using Apache Kafka and Apache Spark Streaming to detect any real-time threats, in order to stop the hacking.

In part A of the assignment, we have already investigated the open data and developed the machine learning models. **In this part B**, we would need to create a proof-of-concept streaming application to demonstrate the integration of machine learning model, Kafka and Spark streaming and create visualisation for monitoring purpose.

What you are provided with

- Two data files:
  - Streaming_Linux_process.csv
  - Streaming_Linux_memory.csv
- Two model files in zip format
  - Process_model.zip
  - Memory_model.zip
- A Metadata file is included which contains the information about the streaming data
  - Linux Features-Description_PartB.pdf
- These files are available in Moodle in the Assessment section

Information on Dataset

Two streaming data files are provided, which captures information relating to the Linux system's process activity and memory activity. The machine information is also provided, which captures information relating to Linux machines.

The streaming data files are an adapted version of process and memory data from the Internet of Things dataset collected by Dr. Nour Moustafa, using IoT devices, Linux systems, Windows systems, and network devices. For more detailed information on the dataset, please refer to the Metadata file included in the assignment dataset or from the website below https://ieee-dataport.org/documents/toniot-datasets.

In the third part of the assignment i.e. **3. Streaming application using Spark Structured Streaming**, **machine ID** and **sequence of data** is also added for each memory activity and the process activity data, while the columns of "attack" and "type" are not included

- Streaming data created from files "**Streaming_Linux_process.csv**" and "**Streaming_Linux_memory.csv**" would be used to simulate the process and memory data streamed from cluster 1 for machines with ID of **4, 5, 6, 7, 8**

What you need to achieve

The StopHacking company requires a proof-of-concept application for monitoring and detecting the attack in process and memory event data. To achieve this, you need to simulate the streaming data production using Kafka, then show some basic counts after consuming the data, and finally build a streaming application that integrates the machine learning model (provided to you)[1] that can predict the attack to monitor the situation.

A compulsory interview would also be arranged in Week 12 after the submission to discuss your proof-of-concept application.

[1] The provided model is a generic model which uses all columns except "ts", "machine", "sequence"

Architecture

The overall architecture of the assignment setup is represented by the following figure.

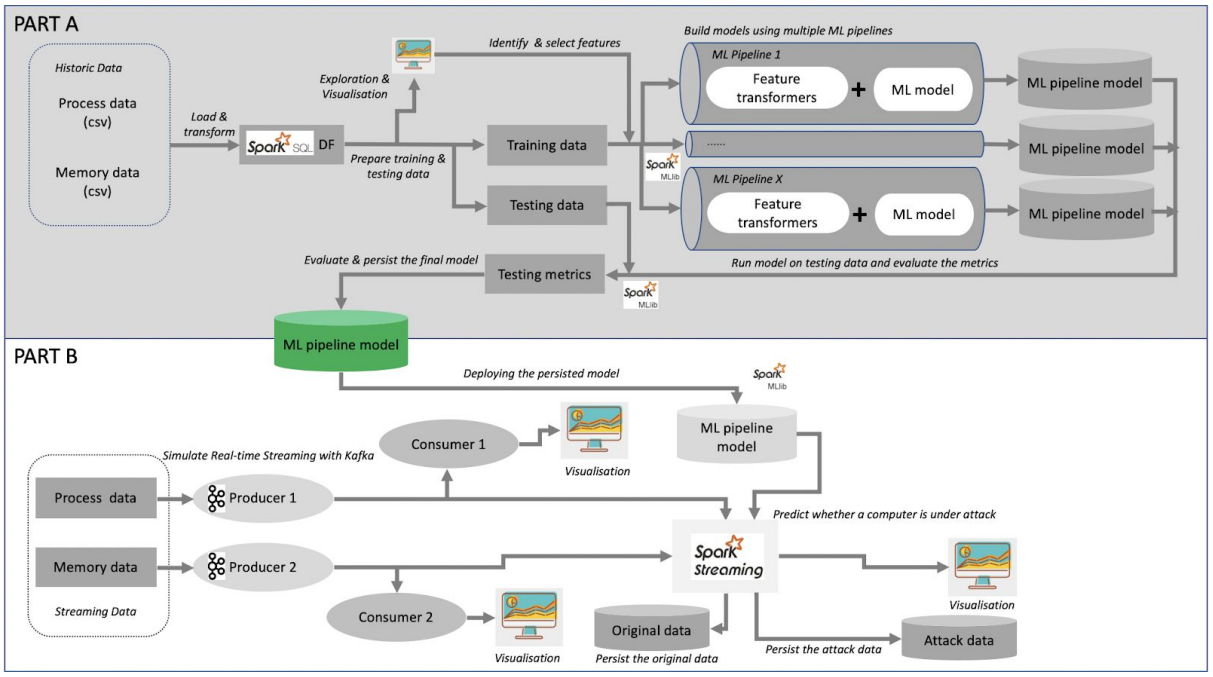

Fig 1: Overall architecture for assignment 2 (part B components updated)

In Part B of the assignment 2, you have two main tasks - producing streaming data and processing the streaming data.

1. In task 1 for producing the streaming data, you can use csv module or Pandas library or other libraries to read and publish the data to the Kafka stream.
2. In task 2 for consuming the streaming using Kafka consumer, you can use csv module or Pandas library or other libraries to process the ingested data from Kafka.
3. In task 3 for streaming data application, you need to use Spark Structured Streaming together with Spark ML / SQL to process the data streams. For task 3, Pandas can only be used for plotting steps, but excessive usage of Pandas for data processing is discouraged.

Please follow the steps to document the processes and write the codes in Jupyter Notebook.

Getting Started

- Download the data and models from moodle.
- Create an ***Assignment-2B-Task1_process_producer.ipynb*** file for data production
- Create an ***Assignment-2B-Task1_memory_producer.ipynb*** file for data production
- Create an ***Assignment-2B-Task2_process_consumer.ipynb*** file for consuming process data using Kafka
- Create an ***Assignment-2B-Task2_memory_consumer.ipynb*** file for consuming memory data using Kafka
- Create an ***Assignment-2B-Task3_streaming_application.ipynb*** file for consuming and processing data using Spark Structured Streaming

1. Producing the data (30%)

In this task, we will implement **two** Apache Kafka producers (one for process and one for memory) to simulate the real-time streaming of the data.

Important:

- In this task, you need to **generate the event timestamp in UTC timezone for each data record in the producer, and then convert the timestamp to unix-timestamp format (keeping UTC timezone) to simulate the "ts" column**. For example, if the current time is 2020-10-10 10:10:10 UTC, it should be converted to the value of 1602324610, and stored in the "ts" column
- **Do not use Spark in this task**

1.1 Process Event Producer (15%)

Write a python program that loads all the data from "*Streaming_Linux_process.csv*". Save the file as ***Assignment-2B-Task1_process_producer.ipynb***.

Your program should send X number of records from **each machine following the sequence** to the Kafka stream **every 5 seconds**.

- The number X should be a random number between 10~50 (inclusive), which is regenerated for each machine in each cycle.
- You will need to append event time in unix-timestamp format (as mentioned above).
- If the data is exhausted, restart from the first sequence again

1.2 Memory Event Producer (15%)

Write a python program that loads all the data from "*Streaming_Linux_memory.csv*". Save the file as ***Assignment-2B-Task1_memory_producer.ipynb***.

Your program should send X number of records from **each machine following the sequence** to the Kafka stream **every 10 seconds**. Meanwhile, also generate Y number of records with the same timestamp. These Y number of records would be sent after 10 seconds (or the next cycle)[2]. A figure demonstrating the timeline is shown below.

- The number X should be a random number between 20~80 (inclusive), which is regenerated for each machine in each cycle.
- The number Y should be a random number between 0~5 (inclusive), which is regenerated for each machine in each cycle.
- You will need to append event time in unix-timestamp format (as mentioned above).

[2] This is to simulate records that do not arrive in the same cycle.

- If the data is exhausted, restart from the first sequence again

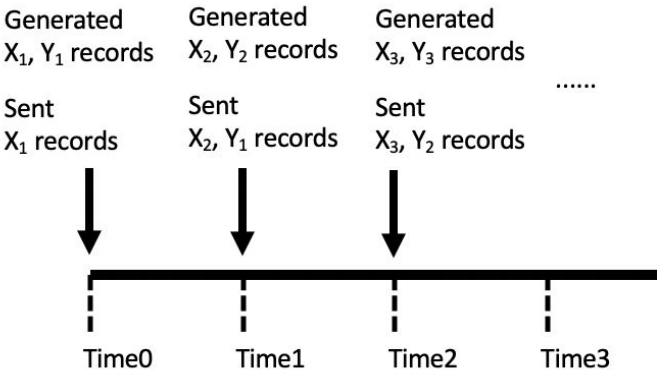

Fig 2: Timeline on the data generation and publication onto the stream

2. Consuming data using Kafka (10%)

In this task, we will implement multiple Apache Kafka consumers to consume the data from task 1.

Important:

- **In this task, use Kafka consumer to consume the data from task 1.**
- **Do not use Spark in this task**

2.1 Process Event Consumer (5%)

Write a python program that consumes the process events using kafka consumer, visualise the record counts in real time. Save the file as ***Assignment-2B-Task2_process_consumer.ipynb***.

Your program should get the count of records arriving in the last 2-minutes (**use the processing time**) for each machine, and use line charts to visualise.

- Hint - x-axis can be used to represent the timeline, while y-axis can be used to represent the count; each machine's line data can be represented in different color legends

2.2 Memory Event Consumer (5%)

Write a python program that consumes the memory events using kafka consumer, visualise the record counts in real time. Save the file as ***Assignment-2B-Task2_memory_consumer.ipynb***.

Your program should get the count of records arriving in the last 2 minutes (**use processing time**) for each machine, and use line charts to visualise the count from the start to the most recent.

- Hint - x-axis can be used to represent the timeline, while y-axis can be used to represent the count; each machine's line data can be represented in different color legends

3. Streaming application using Spark Structured Streaming (60%)

In this task, we will implement Spark Structured Streaming to consume the data from task 1 and perform predictive analytics.

Important:

- **In this task, use Spark Structured Streaming together with Spark SQL and ML**
- **You are also provided with a set of pre-trained pipeline models, one for predicting attack in process data, another for predicting attack in memory data**

Write a python program that achieves the following requirements. Save the file as ***Assignment-2B-Task3_streaming_application.*ipynb**.

1. SparkSession is created using a SparkConf object, which would use two local cores with a proper application name, and use UTC as the timezone[3] (4%)

2. From the Kafka producers in Task 1.1 and 1.2, ingest the streaming data into Spark Streaming for both process and memory activities(3%)

3. Then the streaming data format should be transformed into the proper formats following the metadata file schema for both process and memory, similar to assignment 2A[4] (3%)
    - The numeric values with extra spaces or "K" / "M" / "G" should be properly transformed into their correct values
    - The NICE value should also be restored based on the PRI values using their relationship[5]
        - Hint - There is a mapping between PRI (priority) and NICE, as long as the process is not yet finished during the last interval. For example,
            - PRI 100 maps to NICE -20
            - PRI 101 maps toNICE -19
            - …
            - PRI 139 maps to NICE 19
        - Hint - If the process is finished, PRI and NICE would both be 0.

[3] More details of configuration can be found on https://spark.apache.org/docs/latest/configuration.html
[4] Most columns in assignment 2B are in the same formats of assignment 2A, except for the new columns
[5] Reference from http://www.cs.unc.edu/~porter/courses/comp530/f16/slides/scheduling.pdf

4. For process and memory, respectively, create a new column “CMD_PID” concatenating “CMD” and “PID” columns, and a new column “event_time” as timestamp format based on the unix time in “ts” column (5%)
    - Allow 20-second tolerance for possible data delay on “event_time” using watermarking

5. Persist the transformed streaming data in parquet format for both process and memory (5%)
    - The process data should be stored in “process.parquet” in the same folder of your notebook, and the memory data should be stored in “memory.parquet” in the same folder of your notebook

6. Load the machine learning models given[6], and use the models to predict whether each process or memory streaming record is an attack event, respectively (5%)

7. Using the prediction result, and monitor the data following the requirements below (30%)
    a. If any program in one machine is predicted as an attack in **EITHER** process or memory activity prediction, it could be a false alarm or a potential attack. Keep track of the approximate count of such events **in every 2-min window** for each machine for process and memory, respectively, and write the stream into Spark Memory sink using complete mode[7]
        - Your aggregated result should include machine ID, the time window, and the counts
        - Note that if there are more than one entries having the SAME “CMD_PID” in a 2-min window, get the approximate distinct count
            - For example, if two or more records of “atop” program with the exact same “CMD_PID” are predicted as an attack in the process between 2020-10-10 10:10:10 and 2020-10-10 10:11:09 , only need to count this “atop” program attack once.

    b. If a program in one machine, having the same “CMD” and “PID” in both process and memory streaming data, is predicted as an attack in **BOTH** process and memory activity prediction, then this is considered as an attack event. Find the streaming events fulfilling the criteria, create a new column to record the processing time[8] and persist them in parquet.
        - Note the program with the same “CMD” and “PID” might not be generated at the exact same event time. If the difference between the event times in process and memory is less than 30 seconds and the program fulfills the criteria of matching “CMD” and “PID”, then you should include them for the above checking.

---

[6] The provided model is a generic model which uses all columns except “ts”, “machine”, “sequence”. Make sure you **do NOT drop / rename** any columns before prediction.
[7] Check out different formats of sink on https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
[8] Processing time is usually different from the event time. It can serve other purposes in the further usage of the data, such as auditing.

- If there are multiple entries fulfilling the above criteria in process or memory, do not remove the extra entries
- Persist the program's relevant information (including process & memory data, process & memory's event and processing timestamp, and prediction result) into parquet format with the name of "process_memory_attack.parquet"

8. Visualise the data in line charts for step 7a (5%)

    - For the count of suspect attacks for each machine in step 7a, use Spark SQL to query the data from Spark Memory sink, and prepare a line chart plot for showing the count of suspect attacks for each machine at each 2-min window from the start to the most recent, and refresh the plot every 10 minutes
        - Hint - x-axis can be used to represent the timeline, while y-axis can be used to represent the count; each machine's line data can be represented in different color legends

Assignment Marking

The marking of this assignment is based on quality of work that you have submitted rather than just quantity. The marking starts from zero and goes up based on the tasks you have successfully completed and the quality of the assignment, for example how well the code follows ***programming standards, code documentation, presentation of the assignment notebook, readability of the code, organisation of code*** *and so on*. Please find the PEP 8 -- Style Guide for Python Code here for your reference.

An interview would also be required for demonstrating your knowledge and understanding on the assignment. The interview would be run during week 12 lab, where audio + camera connection is required.

Submission

You should submit your final version of the assignment solution online via Moodle; You must submit the following:

- A zip file of your Assignment 2B folder, named based on your authcate name (e.g. psan002). This should contain
    - **Assignment-2B-Task1_process_producer.ipynb**
    - **Assignment-2B-Task1_memory_producer.ipynb**
    - **Assignment-2B-Task2_process_consumer.ipynb**
    - **Assignment-2B-Task2_memory_consumer.ipynb**
    - **Assignment-2B-Task3_streaming_application.ipynb**

  This should be a ZIP file and *not any other kind of compressed folder (e.g. .rar, .7zip, .tar).* Please do not include the data files in the ZIP file.
- The assignment submission should be uploaded and finalised by Sunday, Nov 1, 2020, 11:55 PM (Local Campus Time).

- Your assignment will be assessed based on the contents of the Assignment 2B folder you have submitted via Moodle. When marking your assignments, we will use the same ubuntu setup as provided to you in Week 01.

Other Information

Where to get help

You can ask questions about the assignment on the Assignments section in the Ed Forum accessible from the on the unit's Moodle Forum page. This is the preferred venue for assignment clarification-type questions. You should check this forum regularly, as the responses of the teaching staff are "official" and can constitute amendments or additions to the assignment specification. Also, you can visit the consultation sessions if the problem and the confusions are still not solved.

Plagiarism and collusion

Plagiarism and collusion are serious academic offences at Monash University. Students must not share their work with any other students. Students should consult the policy linked below for more information.

https://www.monash.edu/students/academic/policies/academic-integrity

See also the video linked on the Moodle page under the Assignment block.

Students involved in collusion or plagiarism will be subject to disciplinary penalties, which can include:

- The work not being assessed
- A zero grade for the unit
- Suspension from the University
- Exclusion from the University

Late submissions

Late Assignments or extensions will not be accepted unless you submit a special consideration form. ALL Special Consideration, including within the semester, is now to be submitted centrally. This means that students MUST submit an online Special Consideration form via Monash Connect. For more details please refer to the **Unit Information** section in Moodle.

There is a **5% penalty per day including weekends** for the late submission.