**TRANSCRIPT**

Enron Corporation's legal team (henceforward referred to as the actor within this presentation) is working on resolving recent press rumors regarding illegal activity within the company. RMRK Consultants are tasked with helping the team in analyzing their internal personnel communication, to identify any individuals that may be behind the alleged illegal activities.

This presentation discusses the design, testing, and suggested implementation of a multi-agent system (MAS) that will aid the legal team in conducting internal digital forensics. Utilizing a sample email dataset, the team can enter suspect names and find out relevant data (such as terms used within the emails, and their frequency; top senders of the emails in question, and provide a match for any particular query). This presentation further discusses elements of the Python code including methods, functions, and libraries.

There are three (3) types of agents that make up the system:

1. CoordinatorAgent

- coordinates the search within the available email dataset

(environment) using the initial evidence provided by the actor (such as

suspect personnel names), and the developing feedback/evidence (such as

resulting query matches, frequently used terms within the emails, and

top senders of the said emails),

- saves and displays the progress and results.

2. SearchAgent

- reads and searches the emails for a match to the initial evidence,

3. AnalysisAgent

- analyzes the discovered data and reports back to the coordinator. The AnalysisAgent class has also been added as a placeholder, should the actor want further analysis logic added.

Let’s take a look at the code. Module re is imported to help with email extraction, threading is imported for running multiple threads in the system in parallel. This will prove to be useful during the SearchAgent discussion. Module os enables us to operate in an OS environment, and certain NLP functionalities require various modules that will be discussed in a moment. The actor will be required to edit the email directory at this point.

Diving into the agent definitions, we now look into CoordinatorAgent class. Its constructor method takes in self (which is the current instance of the class), and the evidence provided by the actor. Empty lists variables are initialized to store vitals generated during the investigation, namely results, frequent terms, and top senders. A sample dataset has been used to test this design (Cohen, 2015). The actor can substitute internal data for further analysis. The coordinate\_search method then extracts only .txt files from the dataset (since it also has .cats files) using the os module. It then loops over each file to extract email content from each .txt file, and add it to one single list (called emails) with delimiters in between.

The next step conducted by this method is to create and start search agent instances (called search\_agents) that will search for evidence (as it loops in each email within the emails attribute) using techniques such as strip to eliminate whitespaces from the emails. This is followed by another for loop that starts execution of each SearchAgent object in the search\_agents list, and it is ensured by the agent.join() method that the main thread waits for all search agents to finish their tasks before proceeding to call the analyze\_results function on the self object.

The analyze\_results method starts by tokenizing each email in the self.results list, and converting it to lowercase. This forms the all\_words attribute, which further is flattened to filter out non-alphanumeric words and stopwords using list comprehension. The stopwords were previously defined using nltk’s function in english. A counter is run to count all occurrences of elements in a list, from which most frequently used words are extracted (up to 5 but this can be edited per the actor’s preference).

Next, extract\_email method is utilized to extract all fields that have the “From:” starting phase. Since the text files in this dataset can result in multiple results from each email, this list is further narrowed to only email addresses using the regex function. Once the list is generated, it is counted to generate two most common sender email addresses and their respective counts (this is also modifiable by the actor).

We now look at the SearchAgent class, which defines its constructor method and the method run() called by the CoordinatorAgent when the SearchAgent thread starts executing. When that happens, search\_email helper method is called to search for evidence in the email content. As discussed previously in the design proposal stage, this method has the ability to deliver user interface prints showing whether an email matched with the evidence, or no evidence was found.

There are modular methods called save\_results and display\_results of the CoordinatorAgent, that serve as placeholders within the code and can be developed further to save results and display them per the actor’s preference.

The code finally has functional testing available, which creates an instance of the CoordinatorAgent class with a sample set of evidence provided by the actor. It calls the coordinate\_search method of the CoordinatorAgent to perform the search based on the provided evidence. As can be seen in the presentation, in using evidence such as “mark” and “compliance” and using the dataset referenced herein, a total of 1702 email files are searched and search agents are activated. The vitals generated that are of interest to the actor are frequent terms and top senders. This functionality can be further unit tested using the code provided.

This MAS system can be very useful in conducting internal forensics as desired by Enron’s legal team, to ensure that the internal personnel held responsible are identified and course-correction can happen.

In summary, this MAS decentralized architecture has been designed in Python to be a digital forensics solution for Enron Corporation’s legal team to help identify company individuals involved in suspected illegal activity. The actor (herein Enron corp) can enter suspect names or categories as evidence (for a particular dataset) within the MAS, which will be used to search emails, and produce frequent terms used within the respective emails, and top senders of those emails. Outputs in the user interface aid in understanding agent activity within the MAS. The README file discusses testing of the system with any particular dataset.