APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 5 No. 2 (2024) 153-173
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

E-Mailyser: Email Forensics with Timeline and Link Analysis

Wong Xian Hui[1], Nurul Hidayah Ab Rahman[1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: hidayahar@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2024.05.02.009

Article Info

Received: 27 July 2024
Accepted: 16 October 2024
Available online: 15 December 2024

Keywords

Forensic Analysis, CSV Email Dataset, Timeline Analysis, Link Analysis

Abstract

Today, the number of tools that can be used to conduct timeline analysis alone on disk images in the market is sufficient. For example, digital forensic tools like Autopsy, Plaso(Log2Timeline), ProDiscover, and FTK Imager. However, it is relatively less common to find digital forensic tools that visualize entities' linkage. Hence, this study proposed E-Mailyser that focuses on the forensic analysis of email datasets with link and timeline analysis approaches. E-Mailyser includes modules such as the main interface module, timeline analysis module, link analysis module, and reporting module. Functional testing results confirm E-Mailyser's ability to create cases, upload and view CSV dataset content, handle errors properly, and forensic analysis modules function successfully in aiding users to construct a timeline of events chronologically and finding relationships between sender and recipient using various visualisation graphs. The user acceptance test shows positive feedback and satisfaction, confirming the effectiveness of E-Mailyser features and performance. To sum up, E-Mailyser demonstrates effective timeline and link analysis, providing clues for unknown connections and understanding user patterns in digital crime investigations.

1.Introduction

In digital forensics, forensic analysis is obtaining factual observations from evidence, forming hypotheses for the evidence collected, and developing meaningful insights and understanding of evidence or crime through hypothesis testing [1]. Hypothesis testing determines the relationship between evidence and a series of events contributing to an incident or a crime [1]. In short, forensic analysis of digital evidence can help forensic investigators to answer investigation questions, such as 'Who' was involved in the case, 'What' has happened, 'Where' did it happen, and 'How' was the crime committed through crime reconstruction, which relies on timeline analysis, link analysis, and functional analysis [1]. Timeline analysis is the reconstruction of events in a timeframe using web artifacts, devices, data, and documents, aiding forensic investigators in understanding user activities in digital crime investigation [2]. Link analysis is a technique that examines digital forensic artifacts to find correlations and reveals unknown relationships and communication patterns between entities or devices [3],[4]. The visual capabilities of link analysis can help forensic investigators to have a clear image of communications between entities involved in a criminal case [4]. This study will integrate timeline and link analysis approaches with data visualization techniques in digital forensics tools to assist users in constructing a chronological timeline of events and finding relationships between sender and recipient in data sources.

However, analysing digital evidence has become more challenging in today's digital world as current forensic analysis tools need more timeline and link analysis approaches to analyse data with visualization. While tools like Autopsy, Plaso, and FTK Imager can conduct timeline analysis, they are less common for visualizing linkage

between entities as limited tools support interactive link analysis in digital forensics. This problem is highlighted as the graphical representation of data is crucial for faster digital evidence identification [5]. Moreover, the gradual increase in the size and data formats of data storage [6] and the increasing cases of cybercrime [7] have brought a rapid rise in the number of digital forensic cases to be investigated and have also significantly increased forensic investigators' workloads. Therefore, this study proposes a digital forensic tool that can be used to analyse a pre-processed CSV email dataset with timeline and link analysis approaches.

The objectives are to design a digital forensics analysis tool specifically designed to work with CSV file datasets, integrating timeline and link analysis approaches to enhance the investigation process. Next, to develop a digital forensics tool, E-Mailyser using Python programming language and Visual Studio Code. Lastly, to test E-Mailyser on the functionality of forensic analysis and user acceptance of the tool.

Datasets from [8] are used as a case study to demonstrate E-Mailyser functionalities. In particular, the E01 disk image is extracted using Autopsy, and a part of the email dataset from the disk image dataset is exported as a CSV file. Subsequently, E-Mailyzer analyzes pre-processed CSV data from the E01 format disk image using link and timeline analysis. In the aspect of system scope, there will be three modules, including "Main Interface," "Analysis," and "Reporting.".

The remaining of the paper is organized as follows: Section 2 discusses the domain of study and the study of existing tools. Section 3 describes the Object-Oriented Software Development methodology used in developing the proposed tool. Next, Section 4 discusses the result and discussion of E-Mailyser, while Section 5 concludes the paper.

2. Related Work

Section 2 discusses the domain of study including digital forensics phases, forensic analysis, and email forensics. A comparison of related existing tools and the proposed tool is also presented.

2.1 Digital Forensics

Digital forensics is a branch of forensic science that uses scientific principles to identify evidence from electronic devices. It uses Locard's Exchange Principle to relate criminals to crimes [9]. However, evidence dynamics in digital forensics prevent direct application of forensic science principles. Digital forensics uses scientific methods to preserve, collect, validate, identify, analyze, interpret, document, and present digital evidence [9]. The rapid development of digital devices has improved forensic capabilities, and crime reconstruction involves five steps: evidence examination, role classification, event reconstruction and testing, event sequencing, and hypothesis testing [5]. Validated tools are required for digital evidence admission in courts.

Digital forensic investigation involves examining digital objects, developing hypotheses, and ensuring the admissibility of digital evidence in a court of law. Digital data, such as files, hard disks, network packets, and memory pages, are discrete collections stored on different mediums. A change in the state of digital objects is called a digital event. An event that violates a policy or law can be described as an incident or a crime. Investigations aim to develop and test hypotheses to answer questions like who was involved, what caused the incident, where it occurred, how, when, and why. Digital evidence, discovered from examinations of digital objects [10], helps determine the existence of an incident.

2.2 Digital Forensic Phases

Digital forensic phases can be categorized into three main phases: data collection, examination, and analysis.

Digital forensic investigations involve data collection to preserve the state of an incident or crime scene [10]. This process involves acquiring digital objects from various mediums, such as video, photography, or handwritten forms, and preserving the collected data to maintain its integrity. Forensic imaging produces forensic images of these objects when the digital device is powered off. However, live acquisition of network log files is an ongoing process as it is not critical. Conclusively, the data collection process involves acquiring digital objects while preserving the state of an incident or crime scene.

The data examination phase involves investigating digital crime scenes and forensic images for possible digital evidence. It consists of four phases: target definition, data extraction and interpretation, data comparison, and knowledge updates [10]. Phase 1 involves searching for evidence from predefined target files. Phase 2 extracts data from the target object, which is then interpreted and compared to original data to ensure integrity. Phase 3 collects extracted data for knowledge updates, checking for overlapping evidence objects and documenting important data for analysis.

The digital forensic investigation involves a five-phase process known as the digital event reconstruction and documentation phase [10]. This phase begins with the evidence examination phase, where digital evidence is analyzed and classified based on class and individual characteristics. Hypotheses are developed based on each evidence object's characteristics, and events are reconstructed and tested. The hypothesis testing phase tests hypotheses formed for an incident. The data analysis process can be carried out in different layers of storage

systems, including physical storage media analysis and network analysis. Physical storage analysis involves the analysis of nonvolatile storage devices such as hard disks [11]. This phase ends when all evidence objects are reconstructed into events, and the focus shifts to data presentation.

2.3 Forensic Analysis

Forensic analysis is a process that uses digital evidence to answer questions related to an incident through hypothesis testing [1]. It involves gathering information and making observations, forming hypotheses, evaluating hypotheses, drawing conclusions, and communicating findings [1]. The first phase involves gathering information and making observations, followed by forming hypotheses and testing them. The second phase involves evaluating hypotheses to determine the relationship between evidence and events contributing to the incident. If the results don't align with the hypotheses, they are revised and tested. The findings are then presented to authorities or courts for decision making. Digital evidence helps answer fundamental questions about incidents and crimes, including sequence, interaction, source evaluation, and attribution [1]. Relational, functional, and temporal analysis are essential for crime reconstruction.

Relational analysis includes the investigation the geographical location of people and computers and communication or transactions between people to reveal a crucial linkage between individuals [1]. Attribution is used to show links to online activities, but it isn't easy to attribute computer activities to a specific individual. Forensic investigators use traditional techniques like stakeouts to reconstruct evidence, forming strong associations between individuals and computer activities.

Functional analysis is a method used to determine the operating system and configuration of a system or application during a crime to identify the source of digital evidence [1]. This involves evaluating embedded information like printer names, directory locations, author's names, and timestamps in documents like Microsoft Office. This information helps forensic investigators associate files with specific computers and reconstruct crime events. Key logs can also be found automatically.

Temporal analysis is a crucial method in forensic investigation, helping to identify patterns and gaps in crime data [1]. It involves creating a timeline to understand events and their relationship with the crime and plotting data in a histogram. Sequencing techniques reveal events using date-time stamps. Traditional methods extract timestamps or MAC timestamps from the file system, while automated software creates a Super Timeline overloaded with information [12]. Despite automated software revolutionizing temporal analysis, manual efforts remain crucial for in-depth forensic analysis [13].

2.4 Email Forensics

Email forensics involves analyzing emails and their contents to verify their authenticity, source, date, time, actual sender, and recipients in a legally sound manner [14]. This discipline aims to provide admissible digital evidence in civil or criminal courts. The field focuses on investigating, extracting, and analyzing emails to gather digital evidence, aiding in solving crimes and incidents forensically soundly. This study integrates timeline and link analysis with visualization graphs to help forensic investigators visualize connections and linkage, reconstruct timelines, and recognize patterns. For example, visualization aids in temporal analysis by creating timelines, facilitating an easy understanding of chronological order, and enhancing pattern identification to identify potential evidence or suspicious activities. Next, linkage analysis visualizes connections between evidence, entities, or events, aiding investigators in understanding the interconnectedness of elements in a crime case. Lastly, visualization also plays a crucial role in data presentation, presenting complex technical findings in a clear and understandable format during legal proceedings, reducing the amount of data displayed to the user, and reducing the bulky display of information [10]. Visualization is essential in modern digital forensics to address challenges and improve forensic investigators' capabilities.

2.5 Study of Existing Tools

In this section, three digital forensic tools were chosen to be explored: Autopsy [15], Plaso [16], and MailXaminer [17]. These digital forensic tools are then compared with each other in terms of their features, such as their analysis focus, user interface, data source, timeline creation feature, linkage creation feature, artifact analysis, operating system support, and availability of the tool.

Table 1 presents comparative summary of existing forensic analysis tools and the proposed tool in terms of their features, such as their analysis focus, user interface, data source, timeline creation feature, linkage creation feature, artifact analysis, operating system support, and availability of the tool. Autopsy, Plaso, and MailXaminer are valuable tools in the digital forensic toolkit, each specializing in different aspects of forensic analysis. Autopsy focuses on comprehensive disk and file system analysis, providing a user-friendly web interface, while Plaso excels in timeline creation, allowing investigators to correlate events across diverse data sources. Moreover, MailXaminer specializes in email forensics, aiding investigators in extracting and analysing evidence from email communications in various data sources.

Table 1 *Comparison table of existing tools.*

Feature	Autopsy[15]	Plaso[16]	MailXaminer[17]	E-Mailyser
Analysis Focus	Disk and file system analysis	Timeline creation and forensic analysis	Email examination and analysis	Email analysis
User Interface	Web-based interface	Command-line interface	GUI-based interface	GUI-based interface
Data Sources	Disk images, local drives, logical files	Various log and forensic data sources, including disk images	Email communication in various formats, web email servers, disk images, messenger	Email communication in CSV format
Timeline Creation Feature	Yes. However, it displays all similar events as a cluster automatically.	Yes. However, it does not visualize the data.	Yes. However, it only displays the total counts.	Yes. It allows users to select attributes to be analyzed and displays the data in visualization graphs with counts.
Linkage Creation Feature	Limited. It provides limited attributes to be visualized.	No. It does not provide any linkage analysis module.	Yes. However, it has only one type of visualization graph.	Yes. It provides a few types of interactive visualization graphs.
Artifact Analysis	File systems, metadata, artifacts	File systems, metadata, artifacts	Email headers, attachments, bodies	Email messages
Operating System Support	Windows, Linux, macOS	Cross-platform	Cross-platform	Cross-platform
Availability of the Tool	Yes (Open source)	Yes (Open source)	No (Proprietary)	Yes (Open source)

3. Methodology

The object-oriented software development model is the methodology used for developing this tool. There are 5 phases, which includes the phases of requirement elicitation, analysis, design, implementation, and testing [18]. The proposed tool, E-Mailyser was developed using software development technique is object-oriented model. The object-oriented model collects information based on domain analysis. In this study, the model consists of 5 phases: the requirement elicitation phase, analysis phase, design phase, implementation phase, and testing phase. The tasks and expected outcomes of each phase of the object-oriented methodology are presented in Table 2.

Table 2 *Software development related tasks and outputs*

Phase	Task	Output
Object-Oriented Requirement Elicitation	- Define project title, problem statements, objectives, and scope. - Conduct meetings with the supervisor to gather initial requirements. - Conduct domain analysis, studying existing forensic tools (Autopsy, Plaso, MailXaminer). - Define features and functions based on existing tools. - Form application model based on acquired requirements.	- Project title, problem statements, objectives, and scope definition. - Meeting notes, initial requirement documentation. - Comparison table, insights into existing tools. - Feature list, functional and nonfunctional requirements. - Application model representation.
Object-Oriented Analysis	- Transform use cases into object models (UML diagrams)	Use case diagrams, activity diagrams, class diagrams, sequence diagrams.

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 153-173

Table 2 *Software development related tasks and outputs (cont).*

Phase	Task	Output
Object-Oriented Design	- Transform the analysis model into a design model. - Define design goals, software architecture, and boundary use cases. - Define Architecture design for E-Mailyser. - Apply Human-Computer Interaction (HCI) design principles. - Define algorithm design for parsing CSV file, timeline analysis, link analsysis etc. - Specify hardware and software platforms for E-Mailyser	- Design model, nonfunctional requirement integration. - Software architecture documentation. - Architecture design for E-Mailyser (Table 3) - Interface design (Table 5), visibility, feedback, and constraints implementation. - Algorithm design documentation (Table 6). - Hardware and software requirements table.
Object-Oriented Implementation	- Translate the solution domain model into an executable representation. - Implement UML diagrams according to the design	- Code implementation, programming languages used (Python). - Executable representation of UML diagrams.
Object-Oriented Testing	- Conduct unit testing and system testing during development. - Conduct functional testing on basic functional modules and forensic functionalities. - Conduct user acceptance testing with experienced users.	- Unit and system test results. - Functional test results, forensic analysis results. - User acceptance test results, feedback from experienced users.

Table 3 *Architectural design for E-Mailyser*

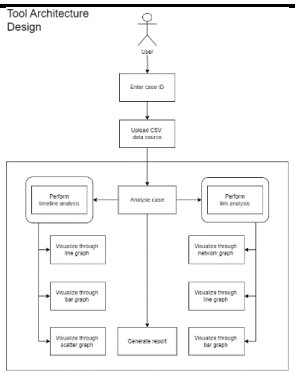
Architectural design	Description
	This figure shows the architecture design of the proposed tool. The proposed tool will have only one actor, which is the user. The user can be an academic researcher or forensic investigator. Firstly, users will be able to create a new case. Next, they will be able to upload a CSV email source to the case created. Then, they can view or edit the case information. Then, users can choose to analyse the CSV email dataset using link and timeline analysis approaches with visualisation graphs to assist in the analysis process. Lastly, users can view the analysis results and generate reports for the case.

Table 4 *E-Mailyser Function Module*

Module	Descriptions
CSV File Upload	Able to upload CSV file dataset.
CSV File Parsing	Able to read and extract CSV file dataset.
Timeline Analysis Module	Provides the capability to create and analyse chronological timelines of events and activities related to the pre-processed email dataset
Link Analysis Module	Identify and visualize relationships between files and entities within pre-processed email dataset
Data Export	Allow users to export analysis graphs and results in various formats (PNG, PDF)

Table 5 *Interface Design*

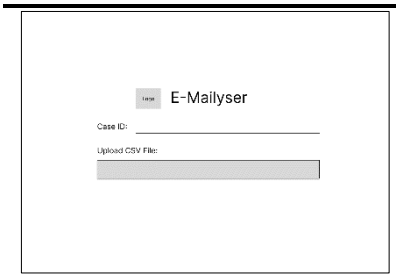
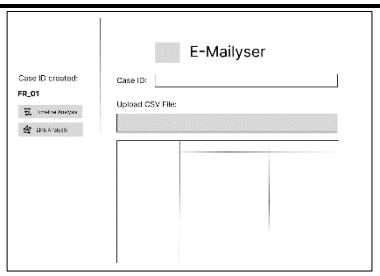
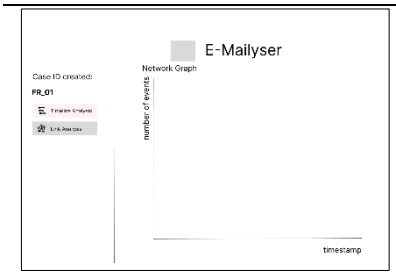
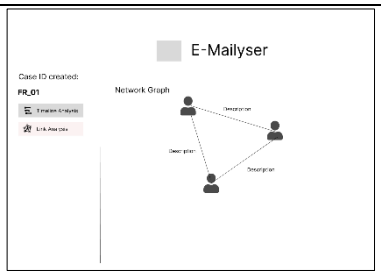
Interface	Description	Interface	Description
	This is the main interface of E-Mailyser, the menu. At this interface, users can create a new case ID and upload CSV file email dataset.		This is the interface design of after a CSV file has been uploaded. The data will be displayed in a table form for users to view the details.
	This is the interface design for the timeline analysis.		This is the interface design for the link analysis.

Table 6 *Algorithm design of the proposed tool*

Algorithm design	Description
If graph_type is 'Line': Display header 'Line Graph' Try: Display description for line graph usage Display info to select "Date Received" as x-axis for timeline Select x_axis from data.columns Select y_axis from data.columns Remove rows with NaN values in [x_axis, y_axis] Sort clean data by [x_axis, y_axis] Call plot_line_chart with clean_data, x_axis, y_axis Catch Exception as e: Display warning to choose different columns with error message	This function creates a line chart from a given DataFrame and selects x and y columns. It also provides options to download the chart and display the relevant data in a table format. Input validation is implemented to check if the specified x_axis and y_axis columns exist in the DataFrame data and if the DataFrame data selected is empty. An error message is displayed if these checks fail, and the function exits early. Next, the next section of the code is used to create a figure and axis using Matplotlib. It plots the data with the x_axis and y_axis with markers at each data point and sets the labels for the x and y-axes and the title of the chart. The x-axis tick labels are rotated by 45 degrees for better readability. Lastly, the plot is displayed using st.pyplot. After the defined function, the user interface will display options for the user to select columns for the x and y axes from the DataFrame by calling the "plot_line_chart" function to create and display the line chart.
Function st_bar_chart_counts(data, x_axis): Count occurrences of each category in x_axis Reset index and rename columns to [x_axis, 'Counts'] Set x_axis as index for plotting If graph_type is 'Bar': Display header 'Bar Graph' Try: Select x_axis from data.columns Remove rows with NaN values in x_axis Count occurrences of each unique value in x_axis and sort by index Sort clean data by x_axis Display markdown 'Counts Plots' Call st_bar_chart_counts with clean_data, x_axis	The code segment is designed to create a bar chart from a given DataFrame based on the counts of unique values in a specified column. The "data[x_axis].value_counts()" counts the occurrences of each unique value in the specified x_axis column while the .reset_index() converts the Series to a DataFrame and resets the index and chart_data.columns = [x_axis, 'Counts'] renames the columns to x_axis and Counts. Next, the index of chart_data is set to the x_axis column and plotted in the chart using Streamlit's st.bar_chart function, which takes the DataFrame chart_data. After that, the graph will be displayed with the table of selected columns below it.

Table 6 *(cont).*

Algorithm design	Description
Function create_network_graph(data, source_column, target_column): Convert source_column and target_column to strings Create graph G from data with source_column and target_column as edges Initialize Network with notebook=True, width="100%", height="800px" Get unique source nodes Get unique target nodes For each node in G.nodes: If node is in sources: Add node to network with color 'blue' and title node Else If node is in targets: Add node to network with color 'green' and title node Else: Add node to network with default color and title node For each edge in G.edges: Add edge to network Create temporary file to save and display network graph Save graph to temporary file Return temporary file name	The code defines a function to create and display a network graph using the NetworkX and Pyvis libraries. The graph visually differentiates source and target nodes with distinct colors. This line "def create_network_graph(data, source_column, target_column):" defines a function named create_network_graph which takes three parameters: data (a DataFrame), source_column, and target_column. Next, the values in the source and target columns will be converted to strings to ensure compatibility with network node identifiers, which may be integers. Then, a NetworkX graph (G) from the DataFrame data from the DataFrame, using the specified source and target columns to define edges with the initialization of a Pyvis Network object (net) to visualize the graph and set the display width and height. Sets of source and target nodes from the data are created using the set keyword. Set is an unordered, unchangeable, and unindexed collection with no duplicate members. The function will then iterate over each node in the NetworkX graph and add it to the Pyvis Network with a specific color: Blue for source nodes, Green for target nodes, and the Default color for other nodes. Next, it adds edges between nodes in the Pyvis Network based on the NetworkX graph's edges. Lastly, it creates a temporary HTML file to save the network graph visualization and returns the path to this temporary HTML file.

4. Results and Discussion

This section presents the results and discussion of the main features of E-Mailyser, which is to assist forensic analysis with timeline analysis and link analysis approaches using various visualization graphs.

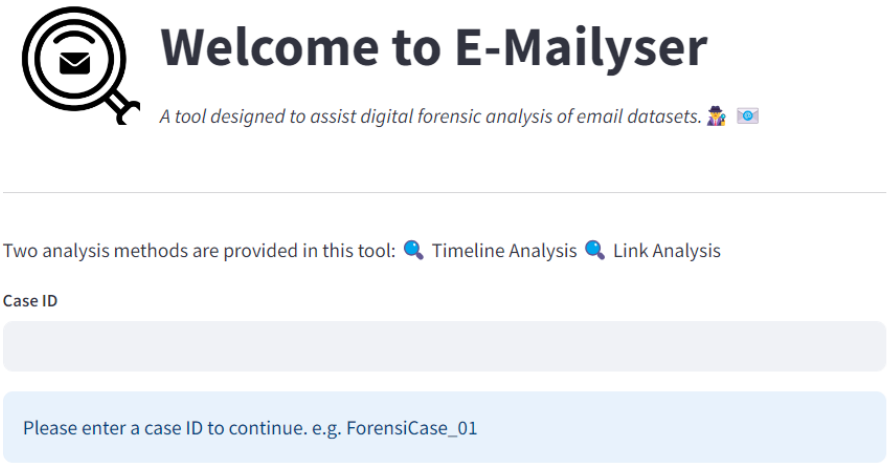


Fig.1 *Interface of the main module*

4.1 Implementation of E-Mailyser

The proposed tool is developed using Python programming language as it provides extensive libraries and can be used to develop online tools on server. Streamlit, an open-source framework that uses Python language to help in displaying data and collecting needed parameters for modeling using graphs. Python libraries such as Pandas, Matplotlib, Altair, Plotly.express, Pyvis, and Networkx are used to read, analyse, and clean data before plotting them on various visualisation graphs.

4.1.1 Implementation of timeline analysis approach

The visualization graphs supported timeline analysis in the tool, including line graphs, bar graphs, and scatter graphs.

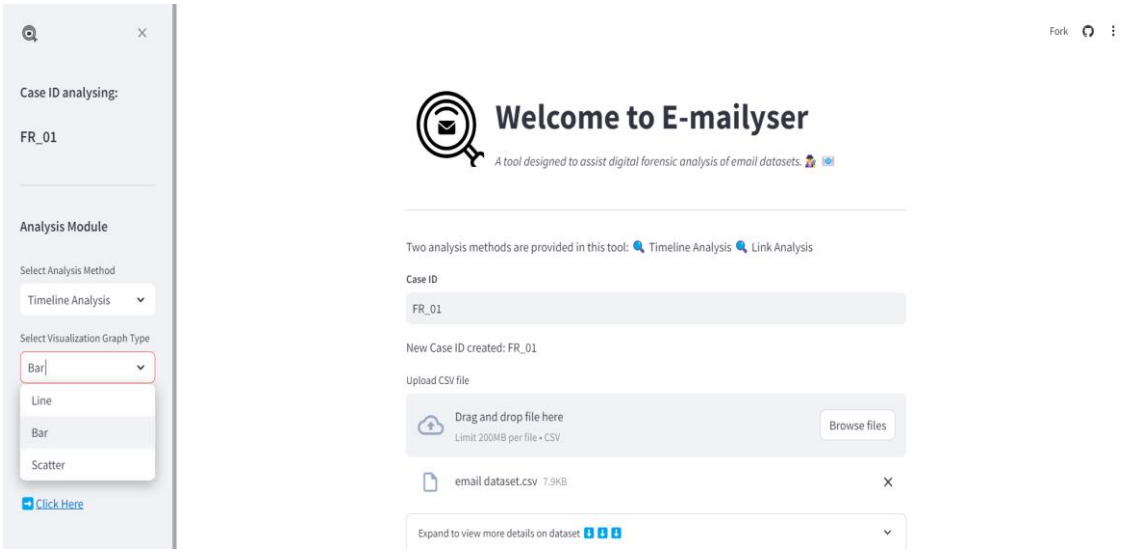


Fig 2 *Timeline analysis module in E-Mailyser*

4.1.1.1 Line Graph

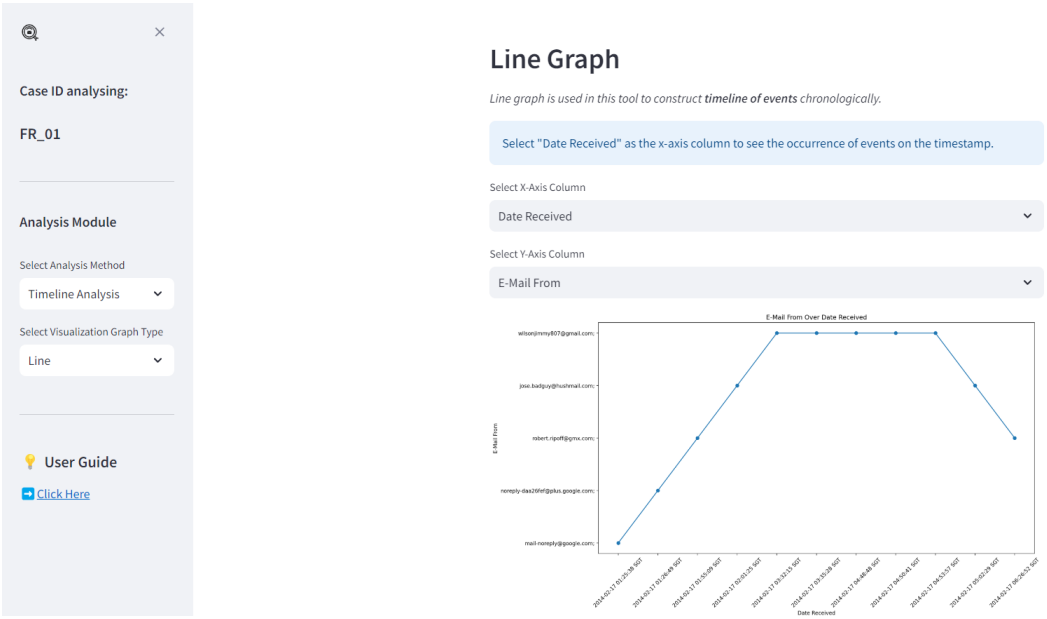


Fig.3 *Line graph interface in the timeline analysis module in E-Mailyser*

Figure 3 shows the line chart displayed in the timeline analysis when the "Date Received" column is chosen as the x-axis while the "E-Mail From" column is chosen as the y-axis. The graph can be read as a user with the email address:mail-noreply@google.com has sent at least one email at the datetime "2014-02-17 01:25:38 SGT". . It helps answer investigation questions such as "What event happened?" and "When did the event occur?" "Who has been involved in the event?" to reveal the sequence of events using date-time stamps [1]. In this case, the

graphical representation of the "Date Received" column and the "E-Mail From" column from the email dataset, we know that there is at least one email sent (What) by the user with the email address:mail-noreply@google.com (Who) on 2014-02-17 01:25:38 SGT (When).

```
# Function to plot line chart
def plot_line_chart(data, x_axis, y_axis):
    if x_axis not in data.columns:
        st.error(f"Column '{x_axis}' not found in the data.")
        return
    if y_axis not in data.columns:
        st.error(f"Column '{y_axis}' not found in the data.")
        return
    if data.empty:
        st.error("The data is empty. Please provide valid data.")
        return

    fig, ax = plt.subplots(figsize=(15, 8))
    ax.plot(data[x_axis], data[y_axis], marker='o')
    ax.set_xlabel(x_axis)
    ax.set_ylabel(y_axis)
    ax.set_title(f'{y_axis} Over {x_axis}')
    plt.xticks(rotation=45)
    st.pyplot(fig)

    buf = fig_to_buffer(plt.gcf())
    st.download_button(label='Download Line Chart', data=buf, file_name='line_chart.png', mime='image/png')

    table_data = data[[x_axis, y_axis]].dropna().reset_index(drop=True)
    table_data.index += 1  # Start the table index from 1
    st.markdown('Table of Selected Columns:')
    st.write(table_data)
```

Fig. 4 *Code segment to plot line graph.*

This function creates a line chart from a given DataFrame and selects x and y columns. It also provides options to download the chart and display the relevant data in a table format. Input validation is implemented to check if the specified x_axis and y_axis columns exist in the DataFrame data and if the DataFrame data selected is empty. An error message is displayed if these checks fail, and the function exits early.

Next, the next section of the code is used to create a figure and axis using Matplotlib. It plots the data with the x_axis and y_axis with markers at each data point and sets the labels for the x and y-axes and the title of the chart. The x-axis tick labels are rotated by 45 degrees for better readability. Lastly, the plot is displayed using st.pyplot. After that, the graph is converted to a buffer using a defined utility function called “fig_to_buffer,” users can download the chart as a PNG file. The selected specified x_axis and y_axis columns from the DataFrame will display the DataFrame as a table below the line graph.

After the defined function, the user interface will display options for the user to select columns for the x and y axes from the DataFrame by calling the “plot_line_chart” function to create and display the line chart. In conclusion, the “plot_line_chart” function in the timeline analysis module validates input columns, plots a line chart using Matplotlib, provides a download button for the chart, and displays the relevant data in a table.

4.1.1.2 Bar Graph

Figure 5 shows the bar chart generated to plot counts in the timeline analysis module. The counts are displayed in a bar chart in the timeline analysis module to determine the periods of highest activity [1]. This graphical representation of the data helps investigators to answer questions: "What and when does an event occur most?". For example, the "Date Received" data column plotted that the highest activity is three emails received, and the periods include "2014-02-17 01:25:38 SGT" and the other date-times listed in the count's table.

Fig 5 *Bar graph interface in the timeline analysis module in E-Mailyser*

```
# Function to plot bar chart
def st_bar_chart_counts(data, x_axis):
   # Count the occurrences of each category in the y-axis column
   chart_data = data[x_axis].value_counts().reset_index()
   chart_data.columns = [x_axis, 'Counts']

   # Set the index to the x-axis column for plotting
   chart_data.set_index(x_axis, inplace=True)

   # Plot the bar chart using Streamlit
   st.bar_chart(chart_data)

# Display bar chart
elif graph_type == 'Bar':
            st.header('Bar Graph')

            try:
               st.write('*Bar graph is used in this tool to plot **counts**.*')
                st.info('Select "Date Received" column to see the occurrence of events on the timestamp. e.g. The
plots will show that there are XX emails received on date XX-XX-XX')
               # Select the x-axis column
               x_axis = st.selectbox('Select a data column to be counted', data.columns)
               # Drop rows with NaN values in the selected column
               clean_data = data.dropna(subset=[x_axis])

               # Count the occurrences of each unique value in the selected column
               email_counts = clean_data[x_axis].value_counts().sort_index()

               clean_data = clean_data.sort_values(by=[x_axis])
               st.markdown('Counts Plots')
               st_bar_chart_counts(clean_data, x_axis)
               # Display the counts as a table
               st.markdown('Counts Table')
               counts_table = pd.DataFrame(email_counts).reset_index()
               counts_table.index += 1  # Add 1 to the index to start from 1
               counts_table.columns = [x_axis, 'Counts']
               st.table(counts_table)

            except Exception as e:
               st.error(f"An error occurred while generating the email counts plot: {e}")
```

Fig.6 *Code segment to plot bar graph.*

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 153-173

Figure 6 shows the code segment designed to create a bar chart from a given DataFrame based on the counts of unique values in a specified column. The "data[x_axis].value_counts()" counts the occurrences of each unique value in the specified x_axis column while the .reset_index() converts the Series to a DataFrame and resets the index and chart_data.columns = [x_axis, 'Counts'] renames the columns to x_axis and Counts. Next, the index of chart_data is set to the x_axis column and plotted in the chart using Streamlit's st.bar_chart function, which takes the DataFrame chart_data. After that, the graph will be displayed with the table of selected columns below it.

4.1.1.3 Scatter Graph

Figure 7 is a scatter chart showing the senders and recipients of emails, the counts, and the timestamp of these events occurring. This graph is designed as a combination of the line and bar graphs provided previously, as it provides information on timestamps and counts in one graph. For example, in the graph, two columns are selected to be plotted as y axes: "E-Mail From" and "E-Mail To," while the chosen index column is "Date Received." Hence, we can see from the graph that the user mailnoreply@google.com has sent three emails to wilsonjimmy807@gmail.com at the time of "2014-02-17 01:25:38 SGT". It is helpful for investigation, such as finding the relationship between the senders and recipients (Who) within a period (When) and the events that occurred between them (What) [17]

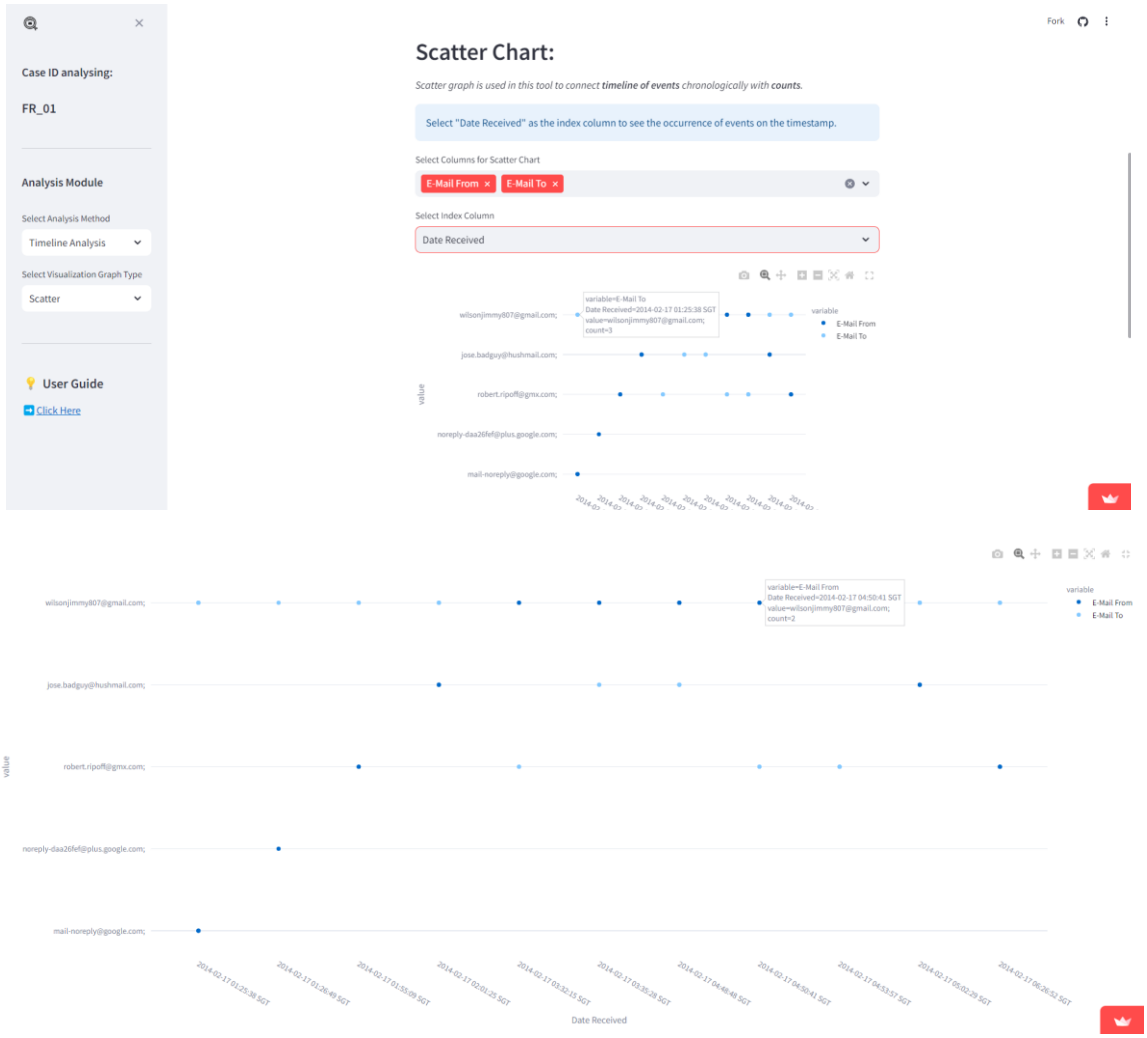


Fig.7 *Scatter graph interface in the timeline analysis module in E-Mailyser*

```
# Function to plot the scatter chart
def st_scatter_chart(data):
   try:
      st.info('Select "Date Received" as the index column to see the occurrence of events on the timestamp.')

      # Allow multiple select for both axes and index_column
      axes = st.multiselect('Select Columns for Scatter Chart', data.columns)
      index_column = st.selectbox('Select Index Column', data.columns)

      if not axes or not index_column:
         st.warning("Please select at least one column for both axes and index.")
      else:
         scatter_data = data.set_index(index_column)[axes].reset_index()
         scatter_data['count'] = scatter_data.groupby(index_column).transform('count').iloc[:, 0]

         # Sort scatter_data based on index_column
         scatter_data.sort_values(by=index_column, inplace=True)

         fig = px.scatter(scatter_data, x=index_column, y=axes, hover_data={'count': True})
         st.plotly_chart(fig)
         st.write('*Hover over the plots to view counts*')


         # Display a table based on the selected index column and y-axis columns
         table_data = scatter_data
         table_data.index += 1  # Start the table index from 1
         st.markdown('Table of Selected Columns:')
         st.write(table_data)
   except Exception as e:
      st.error(f"An error occurred while generating the graph: {e}")

# Display the scatter chart
    elif graph_type == 'Scatter':
               st.header('Scatter Chart:')
                 st.write('*Scatter graph is used  in this tool to connect **timeline of events** chronologically with
**counts**.*')

               try:

                  st_scatter_chart(data)

               except Exception as e:
                  st.error(f"An error occurred while generating the graph: {e}")
```

Fig.8 *Code segment to plot scatter graph.*

Figure 8 shows the code snippet of the function to create a scatter chart from a given DataFrame based on selected columns for the x and y axes and an index column. A multi-select dropdown and a dropdown for selecting the index column are provided for users to plot the columns on the scatter chart axes. A 'count' column that counts occurrences based on the index column is included in the chart displayed. Then, the index column DataFrame will be sorted and plotted Using Plotly Express to create a scatter chart with the index column on the x-axis and the selected columns on the y-axis. Another reason to use Ploty Express is that it enables users to hover over the plots to display counts. The scatter chart in the tool deployed on Streamlit is displayed using st.plotly_chart.

4.1.2 Implementation of link analysis approach

Figure 9 shows the type of visualization graphs supported for link analysis in the tool. The graphs include network graphs, line graphs, and bar graphs.

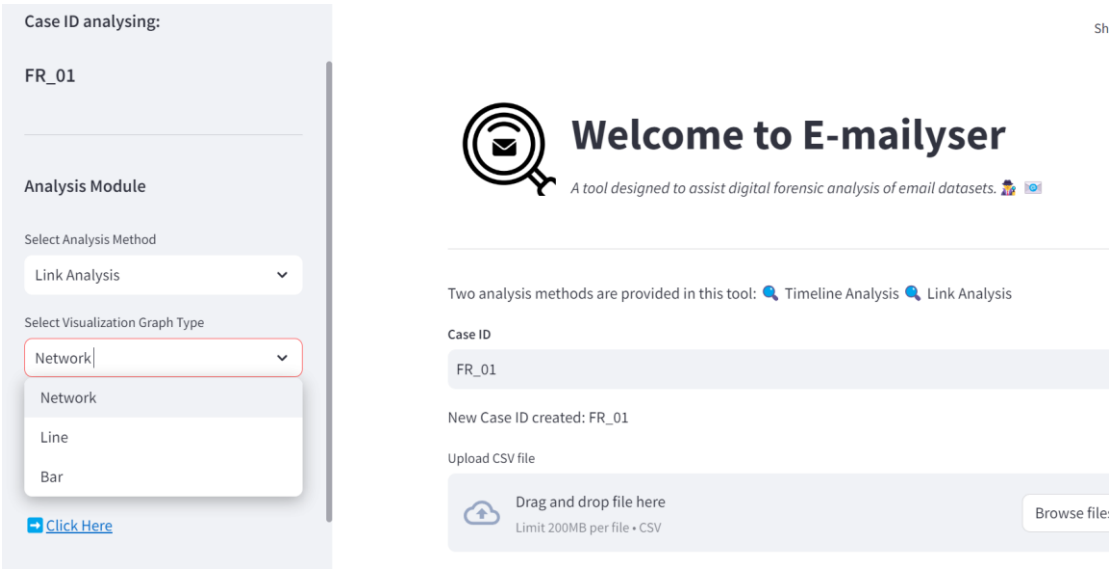


Fig 9 *Link analysis module in E-Mailyser*

4.1.2.1 Network Graph

Figure 10 shows a network graph in the link analysis module that uses the "E-Mail From" and "Source Name" data columns to visually represent communication between users. Attribution is the edges formed between the sources and target nodes to show linkage between users and events [1]. The visualisation graph helps investigators to reveal a crucial linkage between different senders and receivers (Who) and the event done by them (What) [17].

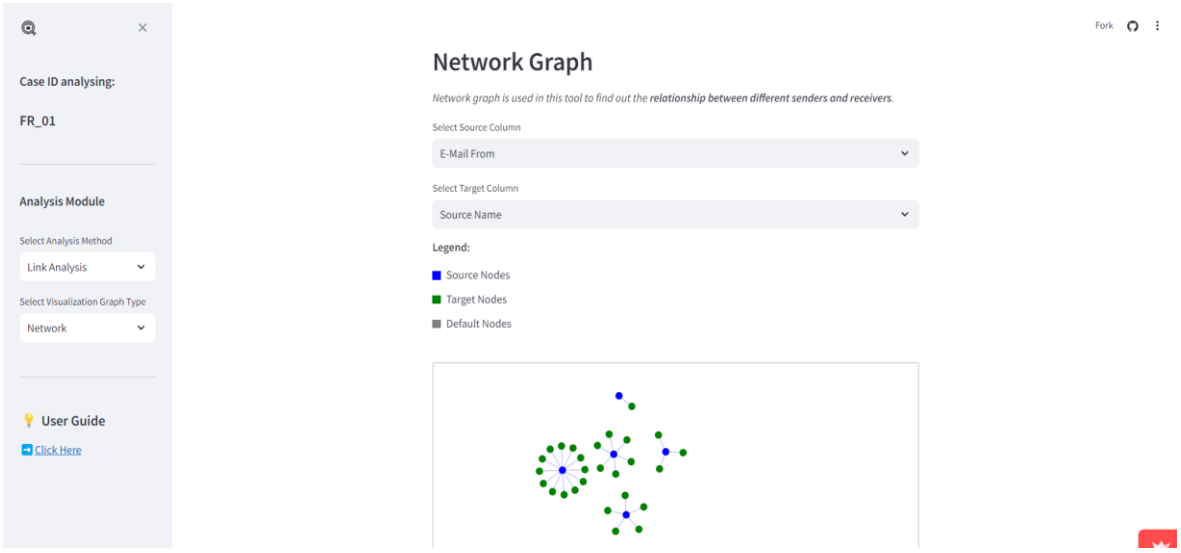


Fig.10 *Network graph interface in the link analysis module in E-Mailyser*

```
# Function to create and display network graph with different colors for source and target nodes
def create_network_graph(data, source_column, target_column):
    # Convert source and target columns to strings to handle integer node identifiers
    data[source_column] = data[source_column].astype(str)
    data[target_column] = data[target_column].astype(str)

    G = nx.from_pandas_edgelist(data, source_column, target_column)

    net = Network(notebook=True, width="100%", height="800px")

    sources = set(data[source_column])
    targets = set(data[target_column])

    # Add nodes with specific colors
    for node in G.nodes:
        if node in sources:
            net.add_node(node, color='blue', title=node)
        elif node in targets:
            net.add_node(node, color='green', title=node)
        else:
            net.add_node(node, title=node)  # Default color for other nodes

    # Add edges
    for edge in G.edges:
        net.add_edge(edge[0], edge[1])

    # Temporary file to save and display the network graph
    with tempfile.NamedTemporaryFile(delete=False, suffix=".html") as tmpfile:
        net.save_graph(tmpfile.name)
        return tmpfile.name

#Display the network graph
if graph_type == 'Network':
                st.header('Network Graph')
                st.write('*Network graph is used in this tool to find out the **relationship between different senders
and receivers**.*')
                source_column = st.selectbox('Select Source Column', data.columns)
                target_column = st.selectbox('Select Target Column', data.columns)

                html_file = create_network_graph(data, source_column, target_column)

                # Create legend with colored patches
                legend_html = """
                <div style="margin-bottom: 2rem;">
                    <h6>Legend:</h6>
                    <div style="display: flex; align-items: center; margin-bottom: 0.5rem;">
                        <div style="width: 12px; height: 12px; background-color: blue; margin-right: 0.5rem;"></div>
                        <span>Source Nodes</span>
                    </div>
                    <div style="display: flex; align-items: center; margin-bottom: 0.5rem;">
                        <div style="width: 12px; height: 12px; background-color: green; margin-right: 0.5rem;"></div>
                        <span>Target Nodes</span>
                    </div>
                    <div style="display: flex; align-items: center;">
                        <div style="width: 12px; height: 12px; background-color: grey; margin-right: 0.5rem;"></div>
                        <span>Default Nodes</span>
                    </div>
                </div>
                """
```

```
st.markdown(legend_html, unsafe_allow_html=True)

# Display the network graph in Streamlit
with open(html_file, 'r', encoding='utf-8') as f:
    html_content = f.read()

components.html(html_content, height=800)
# Clean up temporary file
os.remove(html_file)

st.divider()
# Provide a button for the user to download the HTML file
st.download_button(
    label="Download Network Graph",
    data=html_content,
    file_name="network_graph.html",
    mime="text/html"
)

try:
    # Display the data used for the network graph as a table
    st.markdown('Table of Selected Columns:')
    selected_data = data[[source_column, target_column]]
    data.index += 1  # Start the table index from 1
    st.write(selected_data)
except Exception as e:
    st.warning(f"Choose different columns for x-axis and y-axis to obtain meaningful table result: {e}")
```

Fig.11 *Code segment to plot the network graph.*

Figure 11 shows the code that defines a function to create and display a network graph using the NetworkX and Pyvis libraries. The graph visually differentiates source and target nodes with distinct colors. This line "def create_network_graph(data, source_column, target_column):" defines a function named create_network_graph which takes three parameters: data (a DataFrame), source_column, and target_column.

Next, the values in the source and target columns will be converted to strings to ensure compatibility with network node identifiers, which may be integers. Then, a NetworkX graph (G) from the DataFrame data from the DataFrame, using the specified source and target columns to define edges with the initialization of a Pyvis Network object (net) to visualize the graph and set the display width and height. Sets of source and target nodes from the data are created using the set keyword. Set is an unordered, unchangeable, and unindexed collection with no duplicate members.

The function will then iterate over each node in the NetworkX graph and add it to the Pyvis Network with a specific color: Blue for source nodes, Green for target nodes, and the Default color for other nodes. Next, it adds edges between nodes in the Pyvis Network based on the NetworkX graph's edges. Lastly, it creates a temporary HTML file to save the network graph visualization and returns the path to this temporary HTML file.

The next part of the code shows the block of code that generates and displays a network graph in a Streamlit-based tool. It allows users to select columns representing the source and target of connections and visualise the relationships as a network graph. The dropdown menus allow users to select the source and target columns from the DataFrame. After that, the function create_network_graph is called to generate the network graph HTML file based on the selected source and target columns, while an HTML snippet displays a legend for the network graph, explaining the colors of the nodes (source, target, and default nodes). Lastly, an interactive network graph in HTML form is displayed using Streamlit's components.html.

4.1.2.2 Line Graph

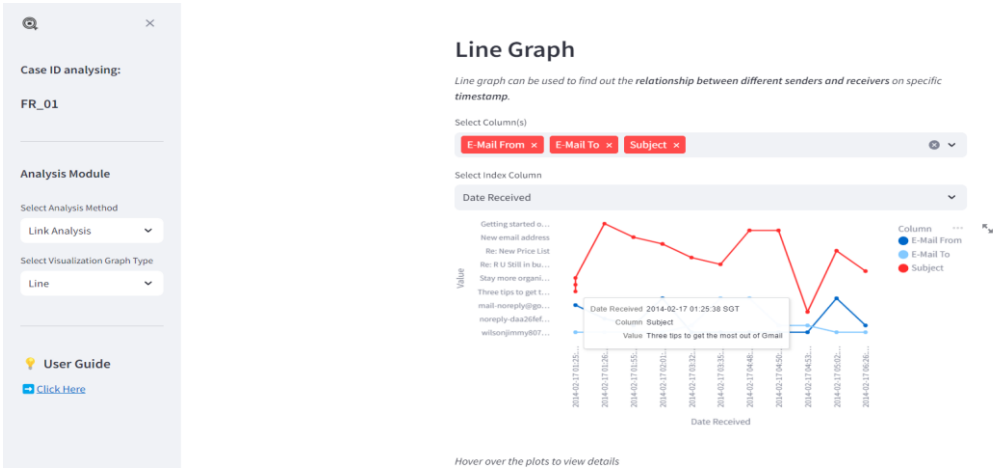


Fig 12 *Line graph interface in the link analysis module in the proposed tool*

Figure 12 shows the link analysis module's line graph to help users visualize the relationship between the email senders and recipients. The subject column is also selected to give information about user connections. The index column "Date Received" is chosen for users to use the timeline constructed to relate to the events between users, in this case, the senders and the recipients. This graph helps answer investigative questions such as "Who has been involved in the occurrence of an event?" "What has occurred between them?" and "When did it happen?". It can prove that there is communication between users over time as link analysis works concurrently with timeline analysis.

```
# Function to display line chart with markers and table
def st_line_chart_with_markers(data, axes, index_column):
    if not axes or index_column in axes:
        st.warning("Selected data columns must not include the index column.")
    else:
        data.set_index(index_column, inplace=True)
        chart_data = data[axes].reset_index().melt(index_column, var_name='Column', value_name='Value')

        line_chart = alt.Chart(chart_data).mark_line(point=True).encode(
            x=index_column,
            y='Value',
            color='Column',
            tooltip=[index_column, 'Column', 'Value']
        ).interactive()
        st.altair_chart(line_chart, use_container_width=True)
        st.write('*Hover over the plots to view details*')
        # Display a table based on the selected index column and y-axis columns
        table_data = data[axes].reset_index()  # Reset index to display the index column in the table
        table_data.index += 1  # Start the table index from 1
        st.markdown('Table of Selected Columns:')
        st.write(table_data)

# Display the line chart
elif graph_type == 'Line':
            st.header('Line Graph')
                st.write('*Line graph can be used to find out the **relationship between different senders and receivers** on specific **timestamp**.*')
            axes = st.multiselect('Select Column(s)', data.columns)
            index_column = st.selectbox('Select Index Column', data.columns)
            clean_data = data.dropna(subset=axes)
            st_line_chart_with_markers(clean_data, axes, index_column)
```

Fig, 13 *Code segment to plot the line graph.*

Figure 13 shows the part of the code that defines a function to display a line chart with markers and a corresponding table. It also handles user interaction to select columns for plotting. The "def st_line_chart_with_markers(data, axes, index_column):" function takes three arguments: data (a DataFrame), axes (list of columns for the y-axis), and index_column (the column for the x-axis). The function ensures that the selected data columns do not include the index column and that at least one column is selected for the y-axis. If this condition is not met, a warning is displayed. The Altair library with the tooltip feature is used to generate an interactive line chart with markers at each data point, and when hovering, details such as the x-axis, y-axis, and column will be displayed. The next section is the user interface where the function will be called to display the plotted line graph with information of the dataset to aid in finding the relationship between the email senders and the recipients.

4.1.2.3 Bar Graph

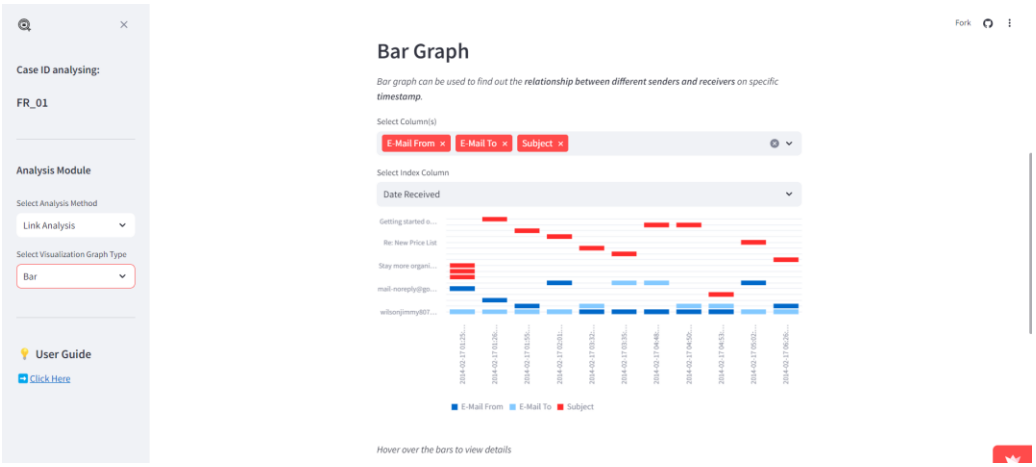


Fig 14 *Bar graph in the link analysis module in the proposed tool*

Figure 14 shows the bar graph displayed in the link analysis module to help users visualizing the relationship between the email senders and recipients. The subject column is also selected to give information about user connections. The index column "Date Received" is chosen for users to use the timeline constructed to relate to the events between users, in this case, the senders and the recipients.

```
# Function to plot bar graph
def st_bar_chart(data, axes, index_column):
   if not axes or index_column in axes:
      st.warning("Selected data columns must not include the index column.")
   else:
   # Set the specified index column
      data.set_index(index_column, inplace=True)

      # Display the line chart using Streamlit
      st.bar_chart(data[axes])
      st.write('*Hover over the bars to view details*')
      table_data = data[axes].reset_index()  # Reset index to display the index column in the table
      table_data.index += 1  # Start the table index from 1
      st.markdown('Table of Selected Columns:')
      st.write(table_data)

# Display the bar graph:
elif graph_type == 'Bar':
            st.header('Bar Graph')
            axes = st.multiselect('Select Column(s)', data.columns)
            index_column = st.selectbox('Select Index Column', data.columns)
            clean_data = data.dropna(subset=axes)

               st_bar_chart(clean_data, axes, index_column)
```

Fig. 15 Code segment to plot the bar chart graph.

Figure 15 shows the part of the code that defines a function to display a bar chart and a corresponding table based on user-selected columns. It also includes the logic to display this bar chart when the user selects the 'Bar' graph type. The function "def st_bar_chart(data, axes, index_column):" takes three arguments: data (a DataFrame), axes (list of columns for the y-axis), and index_column (the column for the x-axis). After retrieving the columns, the bar chart of the selected columns (axes) with the specified index column will be displayed using the st.bar_chart(data[axes]). Table 16 summarizes the functional testing on E-Mailyser. The forensic analysis module: timeline analysis and link analysis aided by visualization graphs are also tested.

Table 7 *Results of Function and Forensic Testing*

Functional Testing Checklist	Pass	Fail
Users can create the case ID.	/	
Users can upload a CSV file extension data source.	/	
Users can view the CSV content in table form.	/	
Users can double-click on a specific cell in the table to view the complete cell content.	/	
The tool can handle errors properly.	/	
Forensic Testing Checklist	**Pass**	**Fail**
The tool should allow the identification of e-mail addresses and internet service provider (ISP) information from the dataset.	/	
Users can select analysis method: timeline analysis or link analysis.	/	
Users can select different types of visualisation graphs for each analysis method.	/	
Users can construct a timeline of events with timestamp using various visualization graphs.	/	
Users can view the counts for each column selected using the bar graph.	/	
Users can find relationships between sender and recipient using various visualization graphs.	/	
The tool should be able to the generate accurate data for timeline analysis.	/	
The tool should be able to the generate complete data for timeline analysis.	/	
The tool should be able to the generate accurate data for link analysis.	/	
The tool should be able to the generate complete data for link analysis.	/	
The tool should be able to generate reports of analysis results for documentation.	/	

Table 7 indicates that E-Mailyser has successfully passed the test plan scenario, specifically in the forensic analysis module.

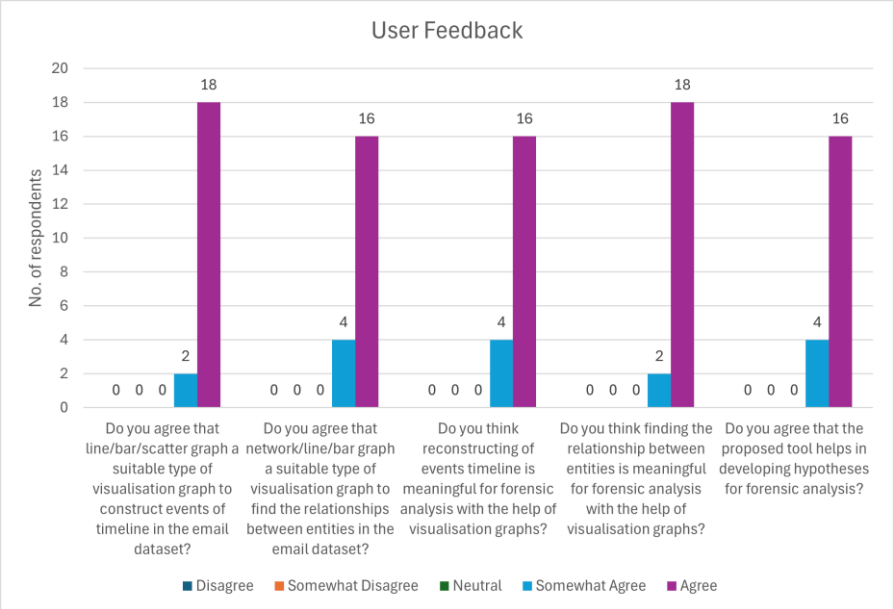


Fig. 16 *Results of User Acceptance Testing on User Feedback*

Figure 16 shows the feedback results after testing the tool functionality and analysis module. Most respondents agree that visualization graphs such as line graphs, bar graphs, and scatter graphs are suitable for constructing timeline events in the timeline analysis module, with only two respondents somewhat agreeing. Next, most of the respondents strongly agree that visualization graphs, such as the network graph, line graph, and bar graph, are suitable for finding relationships between entities in the link analysis module, and four respondents agree. Moreover, 16 respondents strongly agree that visualization graphs help reconstruct a meaningful timeline for forensic analysis, with four agreeing. Next, 18 respondents strongly agreed that finding the relationships between entities is meaningful for forensic analysis using various visualization graphs, and only two respondents agreed. Lastly, most respondents (18) strongly agree that the proposed tool helps develop hypotheses for forensic analysis, and only four agree. The feedback is valuable to evaluate the tool functionality and forensic analysis approaches.

The user acceptance test's positive feedback and high satisfaction confirm the effectiveness of E-Mailyser forensic features with visualisation graphs integrated.

Conclusion

E-Mailyser has been designed, developed, and tested successfully. This section discusses the overall conclusion of the tool, including its advantages, limitations, and suggestions for future works. E-Mailyser has demonstrated its significant in aiding forensic analysis by integrating timeline analysis and link analysis approaches with various visualization graphs. As a result, the implementation can help users in digital forensic investigations and furthermore, speed up the forensic analysis process. However, there are few limitations of the E-Mailyser identified such as:

i. The tool can only allow users to upload one CSV email dataset at a time.
ii. The tool can only allow a maximum file size of 200MB to be uploaded.

To improve the functionality and user experience of the E-Mailyser tool, we recommend the implementation of additional features in the future. These suggestions are as follows:

i. The tool can allow users to upload more than one CSV email datasets at one time.
ii. The tool can integrate Artificial Intelligence (AI) techniques to assist in analyzing large datasets.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The author confirms sole responsibility for the following: study conception and design, data collection, analysis and interpretation of results, and manuscript preparation.

References

[1] E. Casey, *Handbook of Digital Forensics and Investigation*. Elsevier Science, 2009. [Online]. Available: https://books.google.com.my/books?id=xNjsDprqtUYC

[2] S. Bhandari, "Research and implementation of timeline analysis method for digital forensics evidence," Kaunas University of Technology, 2022.

[3] H. Henseler and J. Hyde, "Technology assisted analysis of timeline and connections in digital forensic investigations," *CEUR Workshop Proc.*, vol. 2484, pp. 32–37, 2019.

[4] Y. Ben-moshe, "Joining the forensic dots." Accessed: Jul. 15, 2024. [Online]. Available: https://www.intersecmag.co.uk/forensics-july15/

[5] R. P. Montgomery, "Automated Reconstructions for the Digital Forensic Examiner Workflow," Air Force Institute of Technology, 2022. [Online]. Available: https://scholar.afit.edu/etd/5324

[6] K. Woods, C. A. Lee, and S. Garfinkel, "Extending digital repository architectures to support disk image preservation and access," *Proc. ACM/IEEE Jt. Conf. Digit. Libr.*, pp. 57–66, 2011, doi: 10.1145/1998076.1998088.

[7] "MyCERT : Incident Statistics - Reported Incidents based on General Incident Classification Statistics 2024," MyCERT. Accessed: Jun. 12, 2024. [Online]. Available: https://www.mycert.org.my/portal/statistics-content?menu=b75e037d-6ee3-4d11-8169-66677d694932&id=a6b391c5-3445-44b8-8266-1d85a272e9af

[8] "Forensics Image Test image." Accessed: Jan. 18, 2024. [Online]. Available:

https://cfreds.nist.gov/all/DFIR_AB/ForensicsImageTestimage
[9] André Årnes, *Digital forensics*. John Wiley & Sons Inc., Hoboken, NJ, 2018, 2018. [Online]. Available: https://books.google.com.my/books?hl=en&lr=&id=xqNaDwAAQBAJ&oi=fnd&pg=PR15&ots=q78GhjlWbx&sig=vLev7_mJowxYhRRSHOqPmG8kg88&redir_esc=y#v=onepage&q&f=true
[10] B. Carrier and E. Spafford, "An event-based digital forensic investigation framework," *Digit. forensic Res. Work.*, no. January, pp. 1–12, 2004, [Online]. Available: http://www.digital-evidence.org/papers/dfrws_event.pdf
[11] B. Carrier, *File System Forensic Analysis*. Addison-Wesley, 2005. [Online]. Available: https://books.google.com.my/books?id=I4gpAQAAMAAJ
[12] X. Q. Chow and N. H. Ab Rahman, "A Mobile Forensic Visualization Tool For Android Data Partition," *Appl. Inf. Technol. Comput. Sci.*, vol. 2, no. 2, pp. 37–52, 2021.
[13] N. A. Adderley, "Graph-based Temporal Analysis in Digital Forensics," 2019, [Online]. Available: https://scholar.afit.edu/etd/2241
[14] "Email Forensics - Definition and Guideline," Salvation DATA. Accessed: Jun. 12, 2024. [Online]. Available: https://www.salvationdata.com/knowledge/email-forensics-definition-and-guideline/
[15] Autopsy, "Autopsy - Download." [Online]. Available: https://www.autopsy.com/download/
[16] L.Log2timeline, "GitHub - log2timeline/plaso: Super timeline all the things," GitHub. Accessed: Nov. 28, 2023. [Online]. Available: https://github.com/log2timeline/plaso
[17] M.MailXaminer, "Email Forensics Tool - Best Email Investigation Solution for LEAs," Copyright © MailXaminer. All Rights Reserved. Accessed: Nov. 29, 2023. [Online]. Available: https://www.mailxaminer.com/product/
[18] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java™ Third Edition*, vol. 821 LNCS. Pearson, 2010.

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 153-173

Appendix

(a) **(b)**

Fig. 17 *(a) Evidence user acceptance testing (b) Evidence user acceptance testing*