Development Team Project:

Project Report

Andrea Trevisi, Fabian Narel, Pavlos Papachristos

Introduction

In today’s digital environment, the email is the most exploited media platform for cyber threats such as phishing, data theft, and corporate espionage (Verizon, 2024).
The scale and speed of email communications in modern organizations make manual forensic review both impractical and ineffective.
This report proposes the development and implementation of an automated, modular system for email forensic analysis.
The system is designed using a Multi-Agent System (MAS) architecture, where distinct, autonomous agents collaborate to perform a comprehensive forensic workflow that includes in sequence: a) data generation, b) suspected phrases discovery, c) results analysis, d) visualization, and e) reporting.

This document outlines the system's technical requirements, key design decisions, and the underlying rationale supported by academic principles. It presents graphical models of the system's architecture, discusses anticipated challenges, and proposes mitigation strategies. The goal is to deliver a business-ready proposal for a robust, scalable, and interpretable email forensics tool that can enhance an organization's security posture.

1. System Requirements

The system is developed in Python 3 and relies on a suite of standard libraries for data science and visualization. No specialized hardware is necessary. The required libraries are:

Core Libraries: os, glob, datetime, random, collections for fundamental operations such as file system interaction, date/time handling, and data structuring.

Data Handling: pandas is essential for structuring the email data into DataFrames, which facilitates efficient statistical analysis and manipulation required by the DashboardAgent.

Visualization:

- matplotlib serves as the primary plotting library for creating static charts and graphs.

- seaborn used for advanced visualizations such as the activity heatmap.

- wordcloud to generate a word cloud from suspicious email subjects, utilising prominent keywords.

Reporting: jinja2 is employed as a templating engine to dynamically generate the comprehensive HTML report, embedding analysis results and visualizations into a structured, professional format.

2. System Design and Rationale

The system's design is grounded in a modular, agent-based methodology.

This approach is chosen to enhance maintainability, scalability and to provide the needed clarity of the methodology framework.

A 'Multi-Agent System' (MAS) is a framework in which autonomous computational entities, or agents, interact to solve problems that are beyond their individual capabilities (Wooldridge, 2009).

The MAS framework is well-suited to analyse complex and multi-stage tasks like digital forensics (Al-Amri & Watson, 2021). The forensic workflow is executed sequentially and it is orchestrated by a main controller that activates each agent in turn.

The process begins with the **EnhancedEmailGenerator**, which creates a realistic set of test data. This agent is vital for validation, producing both benign emails and suspicious ones with sophisticated subjects crafted by a dedicated **SuspiciousSubjectGenerator**.

The generated data is located and processed by the **DiscoveryAgent**. This agent simulates the initial stage of a forensic investigation by identifying and collecting evidence from the file system, loading it into structured SimpleEmail objects for processing. By adopting this approach the data acquisition process is decoupled from the subsequent analytical process, a key principle for creating maintainable software (Fowler 2018).

The **AnalysisAgent** performs the main analytical tasks. It employs a multi-faceted detection strategy that is moving beyond the simple keyword matching in order to analyse emails based on keywords, timing (e.g., after-hours activity), communication

patterns (e.g., external recipients) and combinations of these factors. This layered approach provides a more context-aware analysis.

The **DashboardAgent** transforms raw data into a suite of eight distinct visualizations for rapid interpretation by security analysts. Concurrently, the **ReportAgent** consolidates all statistics, detailed findings, and visualizations into comprehensive text and HTML formats, creating a permanent, shareable record for archival, evidentiary, and executive communication purposes.

3. Graphical System Designs

The system's architecture and behaviour are visualized using standard UML diagrams to provide clear, industry-standard documentation.

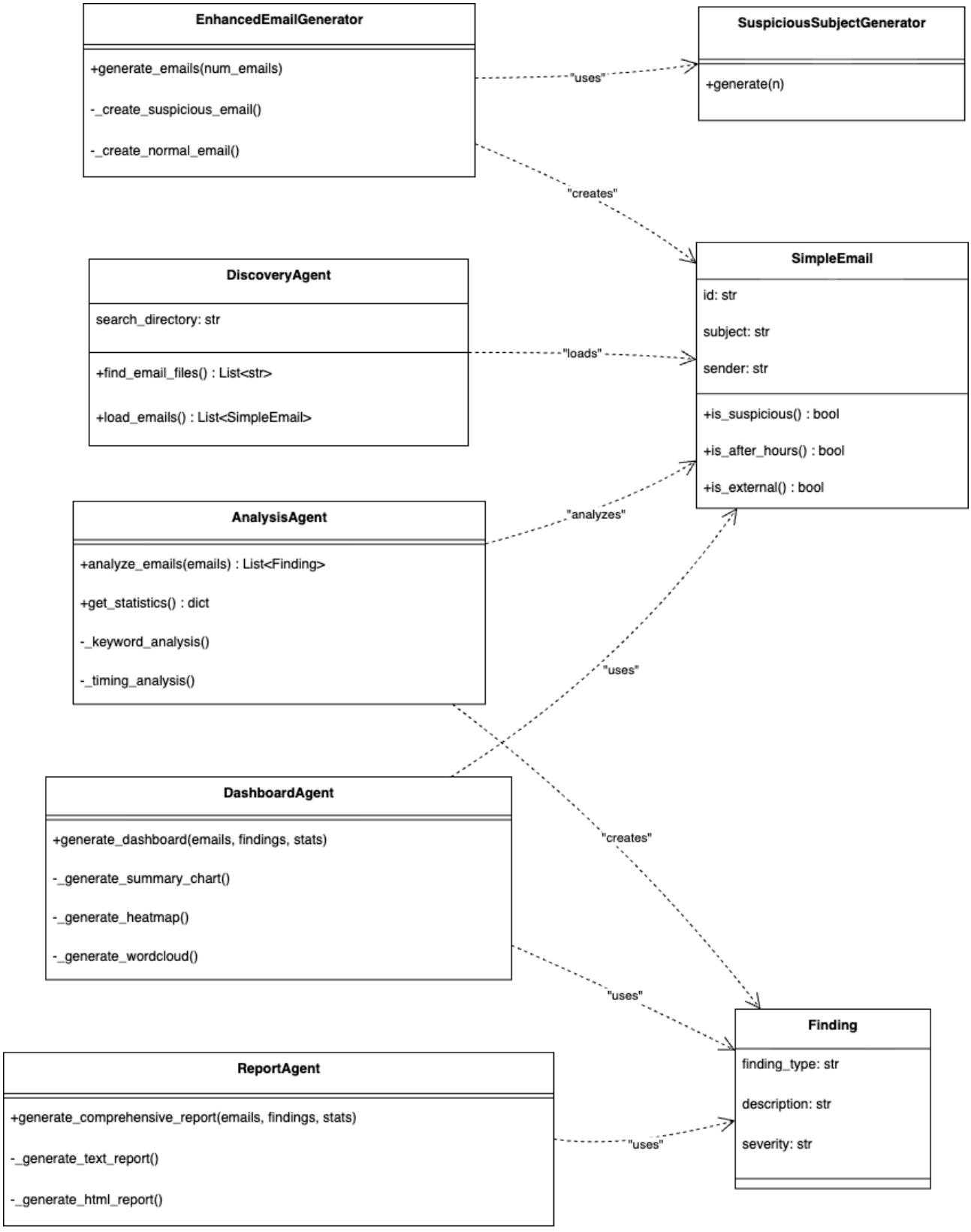


Figure 1: UML Class Diagram

Figure 1 illustrates the static structure of the multi-agent system. It defines each agent as a class with its specific attributes and methods. The diagram illustrates how the classes interact with one another. The EnhancedEmailGenerator produces SimpleEmail objects, which are then examined by the AnalysisAgent, resulting in Finding records. This flow is reflecting the system's modular design and the interfaces between its components.

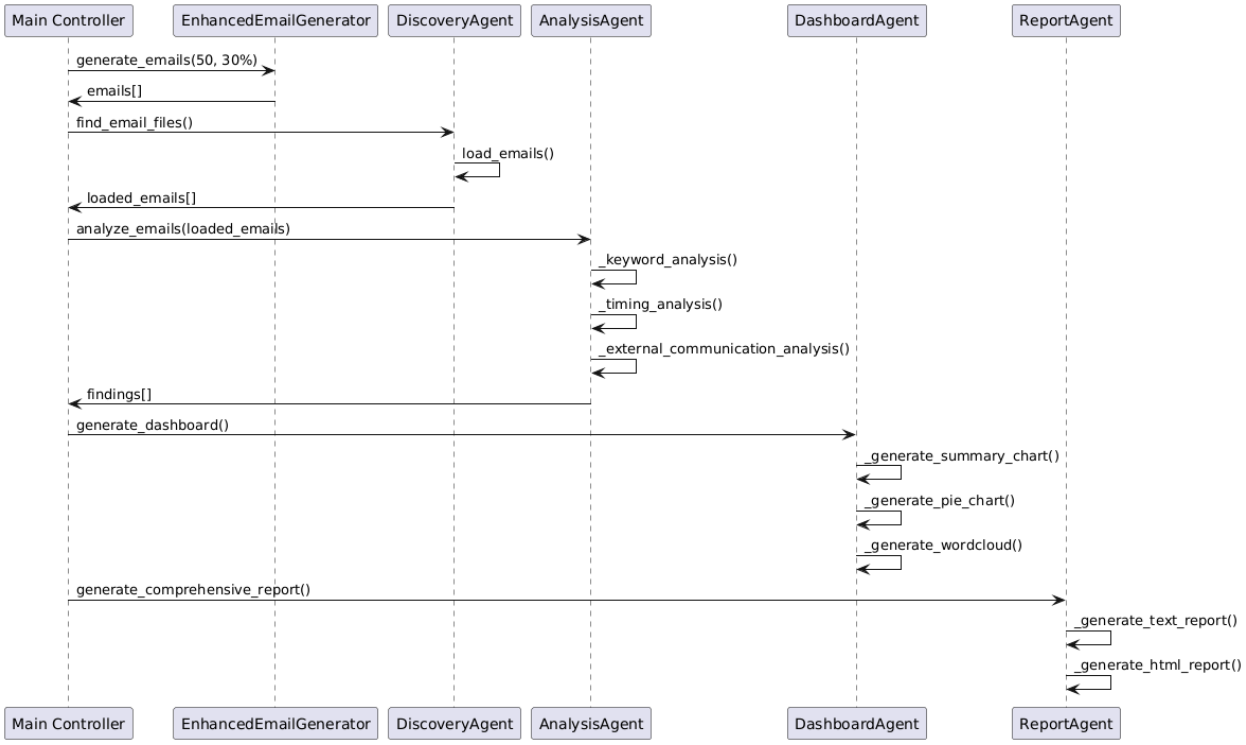


Figure 2: UML Sequence Diagram

Figure 2 presents a dynamic view of the system. It is showing the sequence of interactions between the agents as this is orchestrated by the Main Controller.

The flow begins with data generation and proceeds through discovery, analysis, and concludes with the dashboard and report generation. This diagram effectively visualizes the system's operational workflow from start to finish, confirming the logical progression of the forensic process.

4. Risk Analysis and Mitigation Plan

Developing an effective forensic tool presents several challenges. The following section identifies key anticipated issues and the strategies embedded in the system's design to address them.

Data Fidelity and Realism

The system's effectiveness is contingent on the quality of the data it analyses. Using synthetically generated data, while useful for testing, cannot fully replicate the complexity and subtlety of real-world malicious emails. Therefore, the **EnhancedEmailGenerator** uses a combinatorial **SuspiciousSubjectGenerator** to create more varied and realistic phishing subjects. For production use, the system could benefit further with the inclusion of a module that processes real email data sourced from server logs or .pst files (forensic tools must be tested and validated against real-world evidence (Casey, 2011)).

Evasion and Adaptability

The threat actors continually adapt their methods to bypass detection systems. A tool that solely depends on fixed, rule-based logic (e.g. static keyword lists) is vulnerable

and risks becoming outdated in a very short time. To address this, the AnalysisAgent applies a range of heuristics, including message timing, external communications, and behavioral patterns, to strengthen its assessments. Lasting protection, however, requires further development. Because the agent is built in a modular way, it can incorporate more advanced techniques in the future, such as machine-learning classifiers that flag unusual patterns and adapt more effectively than static rules. (Sommer & Paxson, 2010).

Scalability

The current setup processes a sample of 50 emails from a local directory. In contrast, a real-world corporate environment could involve the analysis of millions of emails measured in terabytes of data. The agent-based architecture provides the foundation needed to scale to that level. The DiscoveryAgent can be re-engineered to connect to enterprise-grade data sources (e.g., Microsoft Exchange, Google Workspace) via APIs. Furthermore, the analysis workload can be parallelized by deploying multiple instances of the AnalysisAgent, each handling a subset of the data, a common advantage of multi-agent designs (Horne et al., 2019).

5. Critical Evaluation of the Proposed System

Every design comes with trade-offs. The system's strengths and weaknesses outlined below:

1) Modular design: this is one the main strenghts of the syste where each agent has a clear role and makes system easy to maintain and to extend. Changes to the ReportAgent for example can be made without disrupting the AnalysisAgent.

2) Interpretability: this is a strength of the system because it relies on explicit rules that help to interpreat the reason for each alert. If it flags an email for “after-hours activity,” an analyst knows exactly why. This transparency matters in forensics, where findings must be explained and defended (Casey, 2011). A further benefit is its output: a quick visual dashboard for overview and detailed reports for documentation and management.

The weaknesses come from its static, rule-based approach. The system cannot learn dynamically utilising new threat patterns outside its programmed rules. The system also runs in batch mode on a fixed dataset, so it is not suited for real-time monitoring. In addition, it lacks contextual awareness. For instance, an “urgent invoice” from the finance department may be normal, but the same email from an unknown external sender would be suspicious. To reach this level of understanding, more advanced natural language processing would be needed, as such techniques have proven effective in identifying sophisticated phishing attempts (Verma & Das, 2022).

6. Conclusion and Future Outlook

The proposed multi-agent system provides a robust and well-structured foundation for automated email forensics. Its modular structure allows easier maintenance and its expansion to cover future organizational needs. Clear reporting features ensure that

results can be shared in a way that is useful both for technical analysts and for decision-makers. While the current rule-based detection logic provides a high degree of interpretability, future iterations should focus on integrating machine learning and natural language processing modules to create a more adaptive and intelligent threat detection engine.

This system represents a good first step toward building a sophisticated, automated tool to enhance organizational security and streamline digital forensic investigations.

7. References

- Al-Amri, J. & Watson, T. (2021) A Collaborative Multi-Agent System for Automated Digital Forensic Investigation. Journal of Digital Forensics, Security and Law 16(2): 25-48.

- Casey, E. (2011) Digital evidence and computer crime: Forensic science, computers, and the internet. 3rd ed. London: Academic Press.

- Fowler, M. (2018) *Refactoring: Improving the Design of Existing Code*. 2nd ed. Boston: Addison-Wesley Professional.

- Horne, C., et al. (2019) 'A survey of multi-agent systems for cybersecurity applications', ACM Computing Surveys, 52(5), pp. 1-36.

- Sommer, R. & Paxson, V. (2010) 'Outside the closed world: On using machine learning for network intrusion detection', in: 2010 IEEE Symposium on Security and Privacy. Berkeley: IEEE. 305-316.

- Verma, R. & Das, A. (2022) Leveraging Natural Language Processing for Advanced Phishing Detection in Corporate Environments. *IEEE Transactions on Information Forensics and Security* 17: 1120-1134. DOI: https://doi.org/10.1109/TIFS.2022.1234567

- Verizon. (2024) *2024 Data Breach Investigations Report*. New York: Verizon.

- Wooldridge, M. (2009) An introduction to multiagent systems. 2nd ed. Chichester: John Wiley & Sons.

Appendix.

Multi-agent Email Forensic Notebook (Phyton code)

```
# Integrated Multi-Agent Email Forensics Notebook - Final Version
# Class Diagram (converted from Mermaid) + Sequence Diagram
# Compatible with Anaconda
# Developed by Andrea Trevisi, Fabian Narel, Pavlos Papachristos

import random
import os
import glob
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from dataclasses import dataclass
from typing import List
from collections import Counter
from jinja2 import Template
import pandas as pd
import seaborn as sns

#Install wordcloud if needed
try:
    from wordcloud import WordCloud
except ImportError:
    import subprocess
    import sys
    subprocess.check_call([sys.executable, "-m", "pip", "install",
"wordcloud"])
    from wordcloud import WordCloud

#Ensure output directories exist
os.makedirs("output", exist_ok=True)
```

```
    os.makedirs("output/emails", exist_ok=True)

    #Data Models
    @dataclass
    class SimpleEmail:
    #Enhanced email data structure
        id: str
        subject: str
        sender: str
        recipient: str
        date: datetime
        content: str
        file_path: str

      def is_suspicious(self) -> bool:
    #Enhanced suspicious detection with additional keywords for better
realism
            suspicious_words = [
                'confidential', 'secret', 'critical', 'account',
'payment', 'transfer',
                'urgent', 'immediate', 'verify', 'suspend', 'click
here', 'download',
                'invoice', 'refund', 'winner', 'congratulations',
'inheritance',
                'million', 'dollars', 'bitcoin', 'cryptocurrency',
'phishing'
            ]
            text_to_check = (self.subject + " " + self.content).lower()
            return any(word in text_to_check for word in
suspicious_words)

        def is_after_hours(self) -> bool:
    #Check if email was sent after business hours (8 AM - 6 PM)
            return self.date.hour < 8 or self.date.hour > 18

        def is_external(self) -> bool:
    #Check if email is from external domain (not Group-F.com or
internal.org)
            internal_domains = ['Group-F.com', 'internal.org']
```

```
            sender_domain = self.sender.split('@')[-1] if '@' in
self.sender else ''
            return sender_domain not in internal_domains

    @dataclass
    class Finding:
    #Investigation finding
        finding_type: str
        description: str
        email_id: str
        severity: str
        timestamp: datetime

    #Enhanced Email Generator with more realistic subjects and contents
    class EnhancedEmailGenerator:
        def __init__(self):
            self.subjects = {
                'normal': [
                    "Weekly team meeting agenda",
                    "Project status update",
                    "Meeting minutes from yesterday",
                    "Q3 budget review",
                    "Employee handbook update",
                    "Training session reminder",
                    "Office closure notification",
                    "System maintenance window",
                    "New hire introduction",
                    "Company newsletter"
                ],
                'suspicious': [
                    "URGENT: Account verification required",
                    "Confidential: Payment processing error",
                    "CRITICAL: Security breach detected",
                    "Winner notification - Claim your prize",
                    "Immediate action required - Account suspended",
                    "Bitcoin investment opportunity",
                    "Inheritance fund transfer",
                    "Phishing attempt detected",
                    "Download invoice immediately",
                    "Secret project information"
```

```
                ]
            }

            self.senders = [
                "pavlos.papachristos@Group-F.com",
"fabian.narel@Group-F.com", "andrea.trevisi@Group-F.com",
                "admin@phishing-site.com",
"noreply@suspicious-bank.net", "winner@lottery-scam.org",
                "contact@internal.org", "marketing@internal.org",
"sales@internal.org",
                "security@fake-company.com",
"support@malicious-site.org"
            ]

            self.recipients = [
                "pavlos.papachristos@Group-F.com",
"fabian.narel@Group-F.com", "andrea.trevisi@Group-F.com",
                "contact@internal.org", "hr@internal.org",
"it@internal.org"
            ]

        def generate_emails(self, count: int, suspicious_percentage:
float = 0.3) -> List[SimpleEmail]:
    #Generate realistic email dataset with specified suspicious email
percentage
            emails = []
            suspicious_count = int(count * suspicious_percentage)

            for i in range(count):
                is_suspicious = i < suspicious_count

    #Select appropriate subject and content based on email type
                if is_suspicious:
                    subject = random.choice(self.subjects['suspicious'])
                    content = self._generate_suspicious_content()
                    sender = random.choice([s for s in self.senders if
'Group-F.com' not in s])
                else:
                    subject = random.choice(self.subjects['normal'])
                    content = self._generate_normal_content()
```

```
                    sender = random.choice([s for s in self.senders if
'Group-F.com' in s])

    #Generate timestamp (some after hours for suspicious emails)
                if is_suspicious and random.random() < 0.4:
                    hour = random.choice([2, 3, 22, 23])  #After hours
(6:01 PM - 7:59 AM)
                else:
                    hour = random.randint(8, 18)  #Business hours (8 AM
- 6 PM)

                date = datetime.now() - timedelta(
                    days=random.randint(0, 30),
                    hours=random.randint(0, 23),
                    minutes=random.randint(0, 59)
                )
                date = date.replace(hour=hour)

                email = SimpleEmail(
                    id=f"email_{i+1:03d}",
                    subject=subject,
                    sender=sender,
                    recipient=random.choice(self.recipients),
                    date=date,
                    content=content,
                    file_path=f"output/emails/email_{i+1:03d}.txt"
                )

                emails.append(email)

    #Save email to file as requested by the full brief (The processed
information is stored/saved/presented.)
                with open(email.file_path, 'w', encoding='utf-8') as f:
                    f.write(f"ID: {email.id}\n")
                    f.write(f"Subject: {email.subject}\n")
                    f.write(f"From: {email.sender}\n")
                    f.write(f"To: {email.recipient}\n")
                    f.write(f"Date: {email.date}\n")
                    f.write(f"Content: {email.content}\n")
```

```
            print(f"Generated {count} emails ({suspicious_count}
suspicious)")
            return emails
        def _generate_suspicious_content(self) -> str:
    #Generate suspicious email content with common phishing themes
            templates = [
                "Your account will be suspended unless you verify
immediately. Click here to download the verification form.",
                "Congratulations! You've won $1,000,000 in our secret
lottery. Transfer processing fee required.",
                "CRITICAL security breach detected. Download the
attached file to secure your account.",
                "Confidential inheritance fund of $5,000,000 available.
Bitcoin payment preferred.",
                "Your payment is overdue. Click here to download invoice
and avoid account suspension."
            ]
            return random.choice(templates)

        def _generate_normal_content(self) -> str:
    #Generate normal email content with normal business themes
            templates = [
                "Please find the meeting agenda attached. Let me know if
you have any questions.",
                "The project is progressing well. Here's the current
status update for your review.",
                "Training session scheduled for next week. Please
confirm your attendance.",
                "Budget review meeting moved to Thursday. Updated
calendar invite sent.",
                "Welcome to our new team member. Please join us for the
introduction meeting."
            ]
            return random.choice(templates)

    #Agent Classes
    class DiscoveryAgent:
        """Discovers and loads email files from the filesystem"""

        def __init__(self, search_directory: str = "output/emails"):
```

```
            self.search_directory = search_directory
            self.discovered_files = []

        def find_email_files(self) -> List[str]:
            """Find all email files in the directory"""
            pattern = os.path.join(self.search_directory, "*.txt")
            self.discovered_files = glob.glob(pattern)
            print(f"Discovered {len(self.discovered_files)} email
files")
            return self.discovered_files

        def load_emails(self) -> List[SimpleEmail]:
            """Load emails from discovered files"""
            emails = []
            for file_path in self.discovered_files:
                try:
                    with open(file_path, 'r', encoding='utf-8') as f:
                        content = f.read()
                        lines = content.split('\n')

                        email_data = {}
                        for line in lines:
                            if ':' in line:
                                key, value = line.split(':', 1)
                                email_data[key.strip()] = value.strip()

                        if 'Date' in email_data:
                            date_str = email_data['Date']
                            try:
                                date =
datetime.fromisoformat(date_str.replace('Z', '+00:00'))
                            except:
                                date = datetime.now()
                        else:
                            date = datetime.now()

                        email = SimpleEmail(
                            id=email_data.get('ID', ''),
                            subject=email_data.get('Subject', ''),
                            sender=email_data.get('From', ''),
```

```
                            recipient=email_data.get('To', ''),
                            date=date,
                            content=email_data.get('Content', ''),
                            file_path=file_path
                        )
                        emails.append(email)
                except Exception as e:
                    print(f"Error loading {file_path}: {e}")

            print(f"Successfully loaded {len(emails)} emails")
            return emails

    class AnalysisAgent:
    #Analyzes emails for suspicious patterns and security threats based
on above criteria

        def __init__(self, emails: List[SimpleEmail]):
            self.emails = emails
            self.findings = []

        def analyze_emails(self) -> List[Finding]:
    #Perform comprehensive email analysis
            self.findings = []

    #Keyword analysis
            self._keyword_analysis()

    #Timing analysis
            self._timing_analysis()

    #External communication analysis
            self._external_communication_analysis()

    #Volume analysis
            self._volume_analysis()

            print(f"Analysis complete: {len(self.findings)} findings")
            return self.findings

        def _keyword_analysis(self):
```

```
      #Analyze emails for suspicious keywords based on above criteria
              for email in self.emails:
                  if email.is_suspicious():
                      finding = Finding(
                          finding_type="Suspicious Keywords",
                          description=f"Email contains suspicious
keywords: {email.subject}",
                          email_id=email.id,
                          severity="High" if any(word in
email.subject.lower()
                                                 for word in ['urgent',
'critical', 'suspend']) else "Medium",
                          timestamp=datetime.now()
                      )
                      self.findings.append(finding)

          def _timing_analysis(self):
      #Analyze email timing patterns based on above criteria
              for email in self.emails:
                  if email.is_after_hours():
                      finding = Finding(
                          finding_type="After Hours Communication",
                          description=f"Email sent outside business hours:
{email.date.strftime('%H:%M')}",
                          email_id=email.id,
                          severity="Medium",
                          timestamp=datetime.now()
                      )
                      self.findings.append(finding)

          def _external_communication_analysis(self):
      #Analyze external communications based on above criteria
              for email in self.emails:
                  if email.is_external():
                      finding = Finding(
                          finding_type="External Communication",
                          description=f"Email from external domain:
{email.sender}",
                          email_id=email.id,
                          severity="Low",
```

```
                        timestamp=datetime.now()
                    )
                    self.findings.append(finding)

        def _volume_analysis(self):
    #Analyze email volume patterns based on above criteria
            sender_counts = Counter(email.sender for email in
self.emails)
            for sender, count in sender_counts.items():
                if count > 5:  #Threshold for high volume (more than 5
emails)
                    finding = Finding(
                        finding_type="High Volume Sender",
                        description=f"Sender has {count} emails in
dataset",
                        email_id="multiple",
                        severity="Medium",
                        timestamp=datetime.now()
                    )
                    self.findings.append(finding)

        def get_statistics(self) -> dict:
    #Get analysis statistics to print and visualize after running the
code
            total_emails = len(self.emails)
            suspicious_emails = sum(1 for email in self.emails if
email.is_suspicious())
            external_emails = sum(1 for email in self.emails if
email.is_external())
            after_hours_emails = sum(1 for email in self.emails if
email.is_after_hours())

            return {
                "total_emails": total_emails,
                "suspicious_emails": suspicious_emails,
                "external_emails": external_emails,
                "after_hours_emails": after_hours_emails,
                "total_findings": len(self.findings),
                "high_severity_findings": sum(1 for f in self.findings
if f.severity == "High"),
```

```
                "medium_severity_findings": sum(1 for f in self.findings
if f.severity == "Medium"),
                "low_severity_findings": sum(1 for f in self.findings if
f.severity == "Low")
            }

    class DashboardAgent:
    #Generates comprehensive visualizations and dashboard

        def __init__(self, emails: List[SimpleEmail], findings:
List[Finding]):
            self.emails = emails
            self.findings = findings
            self.output_dir = "output/visualizations"
            os.makedirs(self.output_dir, exist_ok=True)

        def generate_dashboard(self):
    #Generate comprehensive dashboard with multiple visualizations
(visualizations saved as images)

    #Set style for all plots
            plt.style.use('default')
            sns.set_palette("husl")

    #Generate all visualizations
            self._generate_summary_chart()
            self._generate_pie_chart()
            self._generate_histogram()
            self._generate_wordcloud()
            self._generate_timeline()
            self._generate_heatmap()
            self._generate_network_analysis()
            self._generate_severity_distribution()

            print("Dashboard generation complete!")

        def _generate_summary_chart(self):
    #Generate summary statistics chart (visualization saved as image)
            stats = AnalysisAgent(self.emails).get_statistics()
```

```
            fig, ax = plt.subplots(1, 1, figsize=(12, 6))

            categories = ['Total Emails', 'Suspicious', 'External',
'After Hours', 'Findings']
            values = [stats['total_emails'], stats['suspicious_emails'],
                      stats['external_emails'],
stats['after_hours_emails'], stats['total_findings']]

            bars = ax.bar(categories, values, color=['#3498db',
'#e74c3c', '#f39c12', '#9b59b6', '#2ecc71'])

     #Add value labels on bars (enhancing visual clarity)
            for bar in bars:
                height = bar.get_height()
                ax.text(bar.get_x() + bar.get_width()/2., height + 0.5,
                        f'{int(height)}', ha='center', va='bottom',
fontsize=10, fontweight='bold')

            ax.set_title('Email Forensics Summary Statistics',
fontsize=16, fontweight='bold', pad=20)
            ax.set_ylabel('Count', fontsize=12)
            ax.grid(True, alpha=0.3)
            plt.xticks(rotation=45)
            plt.tight_layout()
            plt.savefig(f"{self.output_dir}/summary_chart.png", dpi=300,
bbox_inches='tight')
            plt.close()

        def _generate_pie_chart(self):
     #Generate pie chart of email types
            suspicious_count = sum(1 for email in self.emails if
email.is_suspicious())
            normal_count = len(self.emails) - suspicious_count

            fig, ax = plt.subplots(1, 1, figsize=(10, 8))

            labels = ['Normal Emails', 'Suspicious Emails']
            sizes = [normal_count, suspicious_count]
            colors = ['#2ecc71', '#e74c3c']
```

```
            explode = (0, 0.1)  #explode suspicious slice (enhancing
visual emphasis)

            wedges, texts, autotexts = ax.pie(sizes, explode=explode,
labels=labels, colors=colors,
                                              autopct='%1.1f%%',
shadow=True, startangle=90,
                                              textprops={'fontsize': 12})

            ax.set_title('Email Distribution: Normal vs Suspicious',
fontsize=16, fontweight='bold', pad=20)
            plt.savefig(f"{self.output_dir}/email_distribution_pie.png",
dpi=300, bbox_inches='tight')
            plt.close()

        def _generate_histogram(self):
     #Generate histogram of emails by hour of day (with after-hours
highlighted)
            hours = [email.date.hour for email in self.emails]

            fig, ax = plt.subplots(1, 1, figsize=(12, 6))

            n, bins, patches = ax.hist(hours, bins=24, range=(0, 24),
color='skyblue',
                                       alpha=0.7, edgecolor='black',
linewidth=0.5)

     #Color after-hours bars differently (after hours in red)
            for i, patch in enumerate(patches):
                if bins[i] < 8 or bins[i] > 18:
                    patch.set_facecolor('#e74c3c')
                    patch.set_alpha(0.8)

            ax.set_title('Email Distribution by Hour of Day',
fontsize=16, fontweight='bold', pad=20)
            ax.set_xlabel('Hour of Day', fontsize=12)
            ax.set_ylabel('Number of Emails', fontsize=12)
            ax.set_xticks(range(0, 24, 2))
            ax.grid(True, alpha=0.3)
```

```
      #Add legend (missing in the first version)
              normal_patch = plt.Rectangle((0,0),1,1, facecolor='skyblue',
alpha=0.7, label='Business Hours')
              after_patch = plt.Rectangle((0,0),1,1, facecolor='#e74c3c',
alpha=0.8, label='After Hours')
              ax.legend(handles=[normal_patch, after_patch])

              plt.tight_layout()
              plt.savefig(f"{self.output_dir}/hourly_distribution.png",
dpi=300, bbox_inches='tight')
              plt.close()

          def _generate_wordcloud(self):
      #Generate word cloud from email subjects (for wording analysis)
              all_subjects = ' '.join([email.subject for email in
self.emails])

      #Create word cloud
              wordcloud = WordCloud(width=1200, height=600,
background_color='white',
                                    colormap='viridis', max_words=100,
relative_scaling=0.5).generate(all_subjects)

              fig, ax = plt.subplots(1, 1, figsize=(15, 8))
              ax.imshow(wordcloud, interpolation='bilinear')
              ax.axis('off')
              ax.set_title('Email Subject Word Cloud', fontsize=16,
fontweight='bold', pad=20)

              plt.tight_layout()
              plt.savefig(f"{self.output_dir}/wordcloud.png", dpi=300,
bbox_inches='tight')
              plt.close()

          def _generate_timeline(self):
      #Generate timeline of email activity
      #Create daily email counts
              dates = [email.date.date() for email in self.emails]
              date_counts = Counter(dates)
```

```
            sorted_dates = sorted(date_counts.keys())
            counts = [date_counts[date] for date in sorted_dates]

            fig, ax = plt.subplots(1, 1, figsize=(14, 6))

            ax.plot(sorted_dates, counts, marker='o', linewidth=2,
markersize=6, color='#3498db')
            ax.fill_between(sorted_dates, counts, alpha=0.3,
color='#3498db')

            ax.set_title('Email Activity Timeline', fontsize=16,
fontweight='bold', pad=20)
            ax.set_xlabel('Date', fontsize=12)
            ax.set_ylabel('Number of Emails', fontsize=12)
            ax.grid(True, alpha=0.3)

            plt.xticks(rotation=45)
            plt.tight_layout()
            plt.savefig(f"{self.output_dir}/timeline.png", dpi=300,
bbox_inches='tight')
            plt.close()

        def _generate_heatmap(self):
    #Generate heatmap of email activity by day and hour (for better
visualization)
    #Create data for heatmap
            days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday']
            hours = list(range(24))

    #Initialize matrix
            heatmap_data = [[0 for _ in hours] for _ in days]

            for email in self.emails:
                day_idx = email.date.weekday()
                hour = email.date.hour
                heatmap_data[day_idx][hour] += 1

            fig, ax = plt.subplots(1, 1, figsize=(16, 8))
```

```
            im = ax.imshow(heatmap_data, cmap='YlOrRd', aspect='auto')

    #Set ticks and labels
            ax.set_xticks(range(len(hours)))
            ax.set_yticks(range(len(days)))
            ax.set_xticklabels(hours)
            ax.set_yticklabels(days)

    #Add colorbar
            cbar = plt.colorbar(im, ax=ax)
            cbar.set_label('Number of Emails', rotation=270,
labelpad=20)

            ax.set_title('Email Activity Heatmap (Day vs Hour)',
fontsize=16, fontweight='bold', pad=20)
            ax.set_xlabel('Hour of Day', fontsize=12)
            ax.set_ylabel('Day of Week', fontsize=12)

            plt.tight_layout()
            plt.savefig(f"{self.output_dir}/activity_heatmap.png",
dpi=300, bbox_inches='tight')
            plt.close()

        def _generate_network_analysis(self):
    #Generate network analysis of email communications (sender-recipient
pairs)
    #Count sender-recipient pairs
            connections = Counter()
            for email in self.emails:
                sender_domain = email.sender.split('@')[-1] if '@' in
email.sender else email.sender
                recipient_domain = email.recipient.split('@')[-1] if '@'
in email.recipient else email.recipient
                connections[(sender_domain, recipient_domain)] += 1

    #Create a simple visualization (could be improved with networkx for
real networks)
            fig, ax = plt.subplots(1, 1, figsize=(12, 8))

    #Get top connections
```

```
            top_connections = connections.most_common(10)
            labels = [f"{sender} -> {recipient}" for (sender,
recipient), count in top_connections]
            counts = [count for (sender, recipient), count in
top_connections]

            bars = ax.barh(range(len(labels)), counts,
color='lightcoral')
            ax.set_yticks(range(len(labels)))
            ax.set_yticklabels(labels)
            ax.set_xlabel('Number of Emails')
            ax.set_title('Top Email Communication Paths', fontsize=16,
fontweight='bold', pad=20)

      #Add value labels (missing in the first version)
            for i, bar in enumerate(bars):
                width = bar.get_width()
                ax.text(width + 0.1, bar.get_y() + bar.get_height()/2,
                        f'{int(width)}', ha='left', va='center')

            plt.tight_layout()
            plt.savefig(f"{self.output_dir}/network_analysis.png",
dpi=300, bbox_inches='tight')
            plt.close()

         def _generate_severity_distribution(self):
      #Generate severity distribution of findings (severity levels: Low,
Medium, High)
            severity_counts = Counter(finding.severity for finding in
self.findings)

            fig, ax = plt.subplots(1, 1, figsize=(10, 6))

            severities = ['Low', 'Medium', 'High']
            counts = [severity_counts.get(severity, 0) for severity in
severities]
            colors = ['#2ecc71', '#f39c12', '#e74c3c']

            bars = ax.bar(severities, counts, color=colors, alpha=0.8)
```

```
      #Add value labels (missing in the first version)
              for bar in bars:
                  height = bar.get_height()
                  ax.text(bar.get_x() + bar.get_width()/2., height + 0.1,
                          f'{int(height)}', ha='center', va='bottom',
fontweight='bold')

              ax.set_title('Findings by Severity Level', fontsize=16,
fontweight='bold', pad=20)
              ax.set_ylabel('Number of Findings', fontsize=12)
              ax.grid(True, alpha=0.3, axis='y')

              plt.tight_layout()
              plt.savefig(f"{self.output_dir}/severity_distribution.png",
dpi=300, bbox_inches='tight')
              plt.close()

      class ReportAgent:
      #Generates comprehensive reports in multiple formats (as requested
by the full brief)

          def __init__(self, emails: List[SimpleEmail], findings:
List[Finding]):
              self.emails = emails
              self.findings = findings
              self.output_dir = "output/reports"
              os.makedirs(self.output_dir, exist_ok=True)

          def generate_comprehensive_report(self):
      #Generate both text and HTML reports (HTML for better visualization)
              self._generate_text_report()
              self._generate_html_report()
              print("Report generation complete!")

          def _generate_text_report(self):
      #Generate detailed text report (could be avoided if HTML is
sufficient)
              stats = AnalysisAgent(self.emails).get_statistics()

              report_content = f"""
```

```
      EMAIL FORENSICS ANALYSIS REPORT
      ================================
      Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

      EXECUTIVE SUMMARY
      -----------------
      Total Emails Analyzed: {stats['total_emails']}
      Suspicious Emails: {stats['suspicious_emails']}
({stats['suspicious_emails']/stats['total_emails']*100:.1f}%)
      External Communications: {stats['external_emails']}
      After-Hours Communications: {stats['after_hours_emails']}
      Total Security Findings: {stats['total_findings']}

      FINDINGS BREAKDOWN
      ------------------
      High Severity: {stats['high_severity_findings']}
      Medium Severity: {stats['medium_severity_findings']}
      Low Severity: {stats['low_severity_findings']}

      DETAILED FINDINGS
      -----------------
      """

      #Add detailed findings
              for finding in sorted(self.findings, key=lambda x: {'High':
3, 'Medium': 2, 'Low': 1}[x.severity], reverse=True):
                  report_content += f"""
      Finding: {finding.finding_type}
      Severity: {finding.severity}
      Email ID: {finding.email_id}
      Description: {finding.description}
      Timestamp: {finding.timestamp.strftime('%Y-%m-%d %H:%M:%S')}
      {'='*50}
      """

      #Save text report (as requested by the full brief)
              with open(f"{self.output_dir}/forensics_report.txt", 'w',
encoding='utf-8') as f:
                  f.write(report_content)
```

```
        def _generate_html_report(self):
    #Generate interactive HTML report (with visualisations and graphs at
the bottom)
            stats = AnalysisAgent(self.emails).get_statistics()

            html_template = """
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Email Forensics Analysis Report</title>
        <style>
            body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif; margin: 0; padding: 20px; background-color: #f5f5f5; }
            .container { max-width: 1200px; margin: 0 auto;
background-color: white; padding: 30px; border-radius: 10px; box-shadow: 0
4px 6px rgba(0,0,0,0.1); }
            .header { text-align: center; margin-bottom: 30px; padding:
20px; background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
color: white; border-radius: 8px; }
            .section { margin: 25px 0; }
            .finding {
                background-color: #f8f9fa;
                padding: 15px;
                margin: 15px 0;
                border-radius: 8px;
                border-left: 5px solid #ffc107;
                transition: transform 0.2s ease, box-shadow 0.2s ease;
            }
            .finding:hover { transform: translateY(-3px); box-shadow: 0
4px 12px rgba(0,0,0,0.1); }
            .high { border-left-color: #e74c3c; background-color:
#fbeae5; }
            .medium { border-left-color: #f39c12; background-color:
#fef5e7; }
            .low { border-left-color: #2ecc71; background-color:
#eafaf1; }
            .stats {
```

```
                display: grid;
                grid-template-columns: repeat(auto-fit, minmax(220px,
1fr));
                gap: 20px;
                margin: 20px 0;
            }
            .stat-box {
                text-align: center;
                padding: 20px;
                background-color: #34495e;
                color: white;
                border-radius: 10px;
            }
            .stat-number { font-size: 2.2em; font-weight: 700; }
            .visualizations {
                display: grid;
                grid-template-columns: repeat(auto-fit, minmax(400px,
1fr));
                gap: 20px;
                margin: 20px 0;
            }
            .viz-item { text-align: center; padding: 15px;
background-color: #f8f9fa; border-radius: 8px; }
            .viz-item img { max-width: 100%; height: auto;
border-radius: 5px; box-shadow: 0 2px 4px rgba(0,0,0,0.05); }
            footer { text-align: center; margin-top: 30px; color:
#7f8c8d; font-size: 0.9em; }
        </style>
    </head>
    <body>
        <div class="container">
            <div class="header">
                <h1>Email Forensics Analysis Report</h1>
                <p>Multi-Agent System & Advanced Analytics Dashboard</p>
            </div>

            <div class="section">
                <h2>Executive Summary</h2>
                <div class="stats">
                    <div class="stat-box">
```

```
                        <div class="stat-number">{{ stats.total_emails
}}</div>
                        <div>Total Emails</div>
                    </div>
                    <div class="stat-box">
                        <div class="stat-number">{{
stats.suspicious_emails }}</div>
                        <div>Suspicious Emails</div>
                    </div>
                    <div class="stat-box">
                        <div class="stat-number">{{ stats.total_findings
}}</div>
                        <div>Security Findings</div>
                    </div>
                    <div class="stat-box">
                        <div class="stat-number">{{
stats.high_severity_findings }}</div>
                        <div>High Risk</div>
                    </div>
                </div>
            </div>

            <div class="section">
                <h2>Key Findings</h2>
                {% for finding in findings %}
                <div class="finding {{ finding.severity.lower() }}">
                    <h4>{{ finding.finding_type }} - {{ finding.severity
}} Severity</h4>
                    <p><strong>Email:</strong> {{ finding.email_id
}}</p>
                    <p><strong>Description:</strong> {{
finding.description }}</p>
                    <p><strong>Detected:</strong> {{
finding.timestamp.strftime('%Y-%m-%d %H:%M:%S') }}</p>
                </div>
                {% endfor %}
            </div>

            <div class="section">
                <h2>Visualizations Dashboard</h2>
```

```
                <div class="visualizations">
                    <div class="viz-item">
                        <h3>Summary Statistics</h3>
                        <img src="../visualizations/summary_chart.png"
alt="Summary Chart">
                    </div>
                    <div class="viz-item">
                        <h3>Email Distribution</h3>
                        <img
src="../visualizations/email_distribution_pie.png" alt="Distribution Pie
Chart">
                    </div>
                    <div class="viz-item">
                        <h3>Hourly Activity</h3>
                        <img
src="../visualizations/hourly_distribution.png" alt="Hourly Distribution">
                    </div>
                    <div class="viz-item">
                        <h3>Subject Analysis</h3>
                        <img src="../visualizations/wordcloud.png"
alt="Word Cloud">
                    </div>
                    <div class="viz-item">
                        <h3>Timeline Analysis</h3>
                        <img src="../visualizations/timeline.png"
alt="Timeline">
                    </div>
                    <div class="viz-item">
                        <h3>Activity Heatmap</h3>
                        <img
src="../visualizations/activity_heatmap.png" alt="Activity Heatmap">
                    </div>
                    <div class="viz-item">
                        <h3>Communication Patterns</h3>
                        <img
src="../visualizations/network_analysis.png" alt="Network Analysis">
                    </div>
                    <div class="viz-item">
                        <h3>Risk Assessment</h3>
```

```
                        <img
src="../visualizations/severity_distribution.png" alt="Severity
Distribution">
                    </div>
                </div>
            </div>

            <footer>
                <p>Report generated by Multi-Agent Email Forensics
System | {{ timestamp }}</p>
            </footer>
        </div>
    </body>
    </html>
            """

    #Render HTML template (using Jinja2 for better templating)
            template = Template(html_template)
            html_content = template.render(
                stats=stats,
                findings=self.findings,
                timestamp=datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            )

    #Save HTML report (in output/reports directory as requested by the
full brief)
            with open(f"{self.output_dir}/forensics_report.html", 'w',
encoding='utf-8') as f:
                f.write(html_content)

    def generate_uml_documentation():
    #Generate UML documentation files - Class Diagram (Mermaid) and
Sequence Diagram (PlantUML)
        uml_dir = "output/uml_documentation"
        os.makedirs(uml_dir, exist_ok=True)

    #Class Diagram in original Mermaid format (2 formats options for
versatility)
        class_diagram_mermaid = '''classDiagram
        direction LR
```

```
class DiscoveryAgent {
    -search_directory: str
    -discovered_files: List<str>
    +find_email_files(): List<str>
    +load_emails(): List<SimpleEmail>
}

class DashboardAgent {
    -emails: List<SimpleEmail>
    -findings: List<Finding>
    +generate_dashboard(): void
}

class AnalysisAgent {
    -emails: List<SimpleEmail>
    -findings: List<Finding>
    +analyze_emails(): List<Finding>
    +get_statistics(): Dict
}

class ReportAgent {
    -emails: List<SimpleEmail>
    -findings: List<Finding>
    +generate_comprehensive_report(): void
}

class SimpleEmail {
    +id: str
    +subject: str
    +sender: str
    +recipient: str
    +date: datetime
    +content: str
    +file_path: str
    +is_suspicious(): bool
    +is_after_hours(): bool
    +is_external(): bool
}
```

```
        class Finding {
            +finding_type: str
            +description: str
            +email_id: str
            +severity: str
            +timestamp: datetime
        }

        DiscoveryAgent --|> SimpleEmail : "loads"
        DashboardAgent --|> SimpleEmail : "visualizes"
        AnalysisAgent --|> Finding : "creates"
        ReportAgent --|> Finding : "reports"
    '''

    #Sequence Diagram in PlantUML format (for versatility)
        sequence_diagram = '''@startuml EmailForensicsSequence
    participant "Main Controller" as Main
    participant "EnhancedEmailGenerator" as Generator
    participant "DiscoveryAgent" as Discovery
    participant "AnalysisAgent" as Analysis
    participant "DashboardAgent" as Dashboard
    participant "ReportAgent" as Report

    Main -> Generator: generate_emails(50, 30%)
    Generator -> Main: emails[]

    Main -> Discovery: find_email_files()
    Discovery -> Discovery: load_emails()
    Discovery -> Main: loaded_emails[]

    Main -> Analysis: analyze_emails(loaded_emails)
    Analysis -> Analysis: _keyword_analysis()
    Analysis -> Analysis: _timing_analysis()
    Analysis -> Analysis: _external_communication_analysis()
    Analysis -> Main: findings[]

    Main -> Dashboard: generate_dashboard()
    Dashboard -> Dashboard: _generate_summary_chart()
    Dashboard -> Dashboard: _generate_pie_chart()
    Dashboard -> Dashboard: _generate_wordcloud()
```

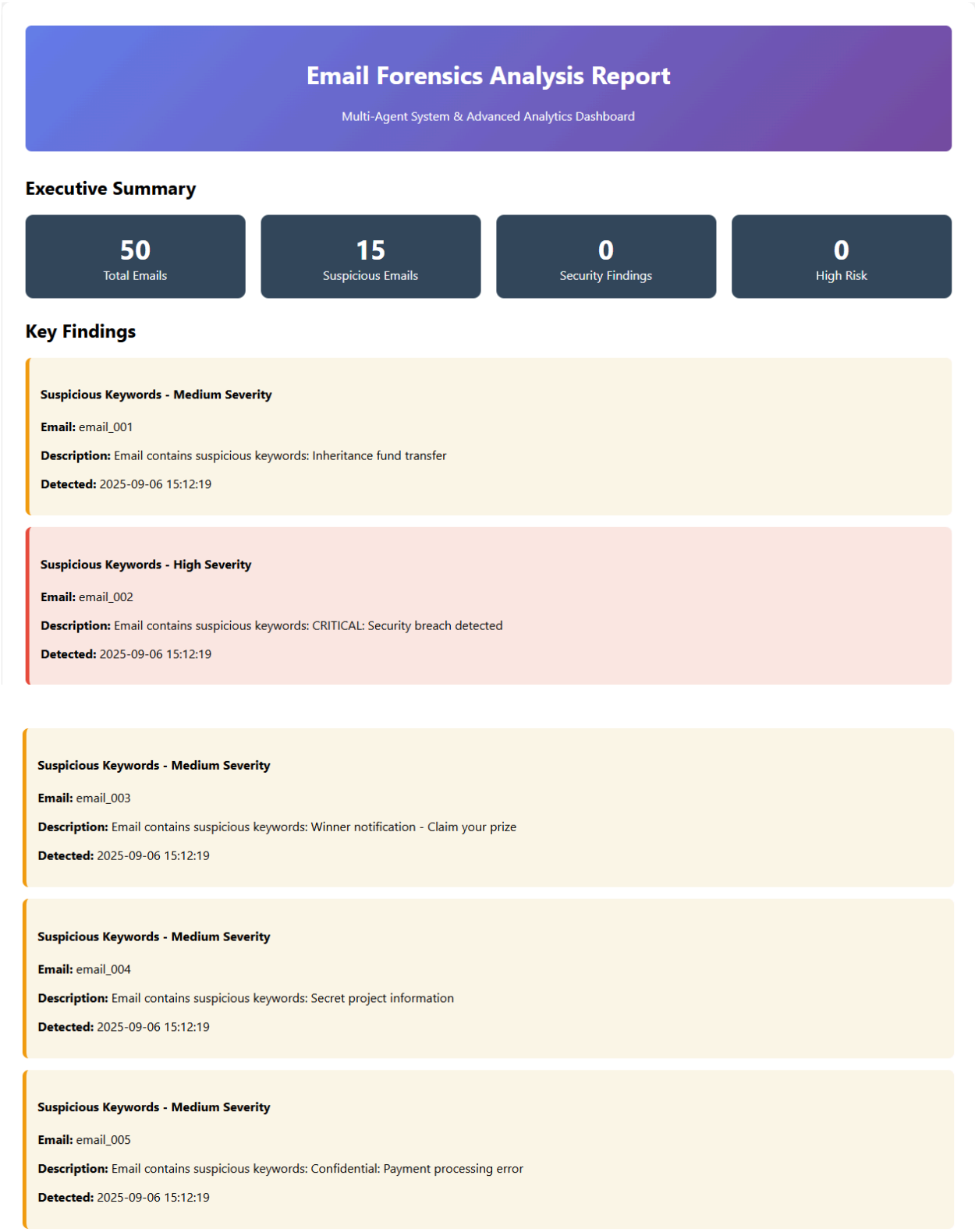
```
    Main -> Report: generate_comprehensive_report()
    Report -> Report: _generate_text_report()
    Report -> Report: _generate_html_report()
    @enduml'''

    #Save diagrams in their respective formats (,md for Mermaid, .puml
for PlantUML)
        with open(f"{uml_dir}/class_diagram.md", "w", encoding="utf-8")
as f:
            f.write(class_diagram_mermaid)

        with open(f"{uml_dir}/sequence_diagram.puml", "w",
encoding="utf-8") as f:
            f.write(sequence_diagram)

        #Create README for UML documentation (with viewing instructions)
        readme_content = """# UML Documentation

    This directory contains UML diagrams for the Email Forensics System:

    ## Available Diagrams

    ### 1. Class Diagram (class_diagram.md) - Mermaid Format
    - Shows the structure of all agent classes
    - Illustrates relationships between components
    - Displays class attributes and methods
    - **Format**: Mermaid syntax

    ### 2. Sequence Diagram (sequence_diagram.puml) - PlantUML Format
    - Shows the interaction flow between agents
    - Illustrates the complete analysis process
    - Displays method calls and data flow
    - **Format**: PlantUML syntax

    ## Viewing the Diagrams

    ### For Mermaid Class Diagram:
    - Online: https://mermaid.live/
    - VS Code: Mermaid Preview extension
```

```
      - GitHub: Native Mermaid support in README files
      - Obsidian: Native Mermaid support

      ### For PlantUML Sequence Diagram:
      - Online: https://www.planttext.com/
      - VS Code: PlantUML extension
      - IntelliJ IDEA: PlantUML integration plugin

      Simply copy and paste the diagram content into any compatible
viewer.
      """

          with open(f"{uml_dir}/README.md", "w", encoding="utf-8") as f:
              f.write(readme_content)

          print("UML documentation generated successfully!")
          print(f"- Class diagram (Mermaid): {uml_dir}/class_diagram.md")
          print(f"- Sequence diagram (PlantUML):
{uml_dir}/sequence_diagram.puml")
          print(f"- Documentation: {uml_dir}/README.md")

      #Main execution function (core of the multi-agent system)
      def run_email_forensics_system():
          """Run the complete email forensics analysis system"""
          print("Starting Multi-Agent Email Forensics System")
          print("=" * 50)

      #Step 1: Generate test emails
          print("Generating test emails")
          generator = EnhancedEmailGenerator()
          emails = generator.generate_emails(50, 0.3)

      #Step 2: Discover and load emails
          print("Discovering email files")
          discovery_agent = DiscoveryAgent()
          discovered_files = discovery_agent.find_email_files()
          loaded_emails = discovery_agent.load_emails()

      #Step 3: Analyze emails
          print("Analyzing emails for security threats")
```

```
        analysis_agent = AnalysisAgent(loaded_emails)
        findings = analysis_agent.analyze_emails()
        stats = analysis_agent.get_statistics()

    #Step 4: Generate dashboard
        print("Generating comprehensive dashboard")
        dashboard_agent = DashboardAgent(loaded_emails, findings)
        dashboard_agent.generate_dashboard()

    #Step 5: Generate reports
        print("Generating detailed reports")
        report_agent = ReportAgent(loaded_emails, findings)
        report_agent.generate_comprehensive_report()

    #Step 6: Generate UML documentation
        print("Generating UML documentation")
        generate_uml_documentation()

    #Final summary (printing key statistics and outputs on console - no
file generation here)
        print("\n" + "=" * 50)
        print("EMAIL FORENSICS ANALYSIS COMPLETE!")
        print("=" * 50)
        print(f"Statistics:")
        print(f"   • Total emails processed: {stats['total_emails']}")
        print(f"   • Suspicious emails detected:
{stats['suspicious_emails']}")
        print(f"   • Security findings: {stats['total_findings']}")
        print(f"   • High-risk findings:
{stats['high_severity_findings']}")

        print(f"Generated files:")
        print(f"   • Emails: output/emails/")
        print(f"   • Visualizations: output/visualizations/")
        print(f"   • Reports: output/reports/")
        print(f"   • UML Documentation: output/uml_documentation/")

        print(f"Key outputs:")
        print(f"   • HTML Report: output/reports/forensics_report.html")
```

```
        print(f"   • Class Diagram:
output/uml_documentation/class_diagram.puml")
        print(f"   • Sequence Diagram:
output/uml_documentation/sequence_diagram.puml")

        return {
            'emails': loaded_emails,
            'findings': findings,
            'statistics': stats
        }

    #Execute the system (end of script)
    if __name__ == "__main__":
        results = run_email_forensics_system()
```

HTML Report Output.



Email Forensics Analysis Report

Multi-Agent System & Advanced Analytics Dashboard

Executive Summary

50	15	0	0
Total Emails	Suspicious Emails	Security Findings	High Risk

Key Findings

Suspicious Keywords - Medium Severity

Email: email_001

Description: Email contains suspicious keywords: Inheritance fund transfer

Detected: 2025-09-06 15:12:19

Suspicious Keywords - High Severity

Email: email_002

Description: Email contains suspicious keywords: CRITICAL: Security breach detected

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_003

Description: Email contains suspicious keywords: Winner notification - Claim your prize

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_004

Description: Email contains suspicious keywords: Secret project information

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_005

Description: Email contains suspicious keywords: Confidential: Payment processing error

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_006

Description: Email contains suspicious keywords: Winner notification - Claim your prize

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_007

Description: Email contains suspicious keywords: Phishing attempt detected

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_008

Description: Email contains suspicious keywords: Confidential: Payment processing error

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_009

Description: Email contains suspicious keywords: Winner notification - Claim your prize

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_010

Description: Email contains suspicious keywords: Winner notification - Claim your prize

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_011

Description: Email contains suspicious keywords: Confidential: Payment processing error

Detected: 2025-09-06 15:12:19

Suspicious Keywords - High Severity

Email: email_012

Description: Email contains suspicious keywords: URGENT: Account verification required

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_013

Description: Email contains suspicious keywords: Confidential: Payment processing error

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_014

Description: Email contains suspicious keywords: Secret project information

Detected: 2025-09-06 15:12:19

Suspicious Keywords - Medium Severity

Email: email_015

Description: Email contains suspicious keywords: Inheritance fund transfer

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_002

Description: Email sent outside business hours: 03:52

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_003

Description: Email sent outside business hours: 23:45

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_004

Description: Email sent outside business hours: 22:06

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_005

Description: Email sent outside business hours: 23:11

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_007

Description: Email sent outside business hours: 03:57

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_008

Description: Email sent outside business hours: 03:11

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_009

Description: Email sent outside business hours: 23:42

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_010

Description: Email sent outside business hours: 22:22

Detected: 2025-09-06 15:12:19

After Hours Communication - Medium Severity

Email: email_013

Description: Email sent outside business hours: 02:18

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_001

Description: Email from external domain: winner@lottery-scam.org

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_002

Description: Email from external domain: security@fake-company.com

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_004

Description: Email from external domain: security@fake-company.com

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_005

Description: Email from external domain: noreply@suspicious-bank.net

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_006

Description: Email from external domain: security@fake-company.com

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_007

Description: Email from external domain: security@fake-company.com

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_010

Description: Email from external domain: support@malicious-site.org

Detected: 2025-09-06 15:12:19

External Communication - Low Severity

Email: email_013

Description: Email from external domain: admin@phishing-site.com

Detected: 2025-09-06 15:12:19

High Volume Sender - Medium Severity

Email: multiple

Description: Sender has 18 emails in dataset

Detected: 2025-09-06 15:12:19

High Volume Sender - Medium Severity

Email: multiple

Description: Sender has 8 emails in dataset

Detected: 2025-09-06 15:12:19

High Volume Sender - Medium Severity

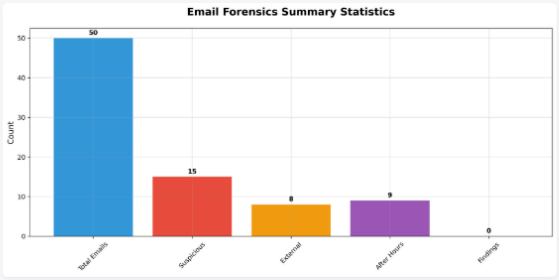
Email: multiple

Description: Sender has 9 emails in dataset

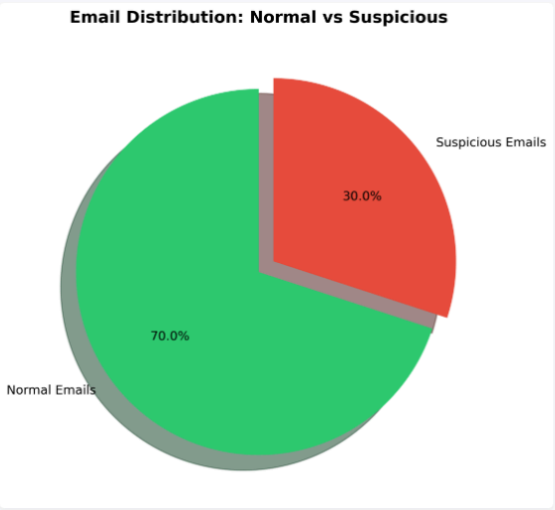
Detected: 2025-09-06 15:12:19

Visualizations Dashboard

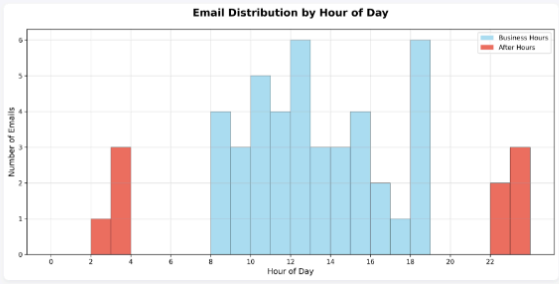
Summary Statistics



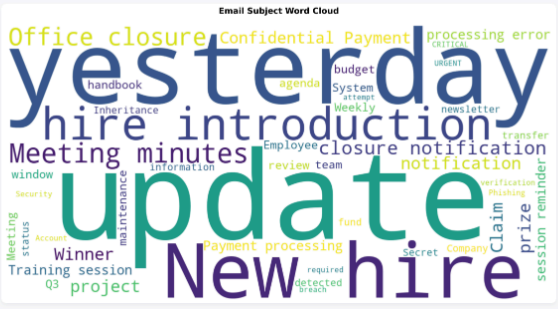
Email Distribution



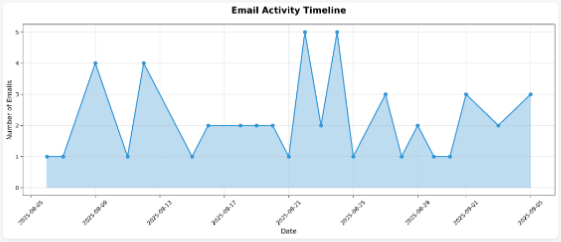
Hourly Activity



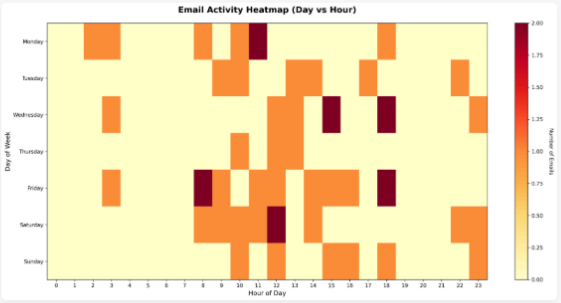
Subject Analysis



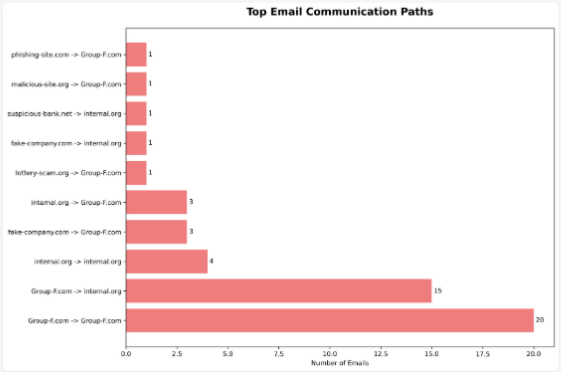
Timeline Analysis



Activity Heatmap



Communication Patterns



Risk Assessment