PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE

ACADEMIC YEAR: 2023-24

DEPARTMENT OF COMPUTER ENGINEERING DEPARTMENT

CLASS: B.E. **SEMESTER: I**

SUBJECT: LP-IV

ASSIGNMENT NO.	A1
TITLE	**Tracking Emails and Investigating Email Crimes. i.e. Write a program to analyze e–mail header**
PROBLEM STATEMENT /DEFINITION	Write a program for Tracking Emails and Investigating Email Crimes. i.e. Write a program to analyze e–mail header
OBJECTIVE	1. Understand the use of Email forensics which is dedicated to investigating, extracting, and analyzing emails 2. understand the Email header analysis important to collect and identify evidence.
OUTCOME	Students will be able to - 1. learn steps to prevent data breaches and Secure email communication. 2. learn to analyze metadata in the email header
S/W PACKAGES AND HARDWARE APPARATUS USED	Windows 10 (64-bit), Intel I5 4GB RAM 256 GB SSD, Python 3.9.0, VS Code
REFERENCES	1.https://commons.erau.edu/cgi/viewcontent.cgi?article=1095&context=jdfsl 2.https://www.youtube.com/watch?v=3wwaYc_Yuhc (Email header analysis tutorial) 3.https://www.tutorialspoint.com/python_digital_forensics/python_digital_forensics_investigation_using_emails.htm
STEPS	Refer to theory, algorithm, test input, test output
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning Outcome 6. Concepts related Theory 7. Algorithm 8. Test cases 9. Conclusion/Analysis

Prerequisites:

Concepts related Theory:

- **Introduction:** Role of Email in Investigation

  Emails play a very important role in business communications and have emerged as one of the

most important applications on internet. They are a convenient mode for sending messages as well as documents, not only from computers but also from other electronic gadgets such as mobile phones and tablets.

The negative side of emails is that criminals may leak important information about their company. Hence, the role of emails in digital forensics has increased in recent years. In digital forensics, emails are considered as crucial evidences and Email Header Analysis has become important to collect evidence during the forensic process.

An investigator has the following goals while performing email forensics −

- To identify the main criminal
- To collect necessary evidences
- To presenting the findings
- To build the case

2. Techniques Used in Email Forensic Investigation

Email forensics is the study of source and content of email as evidence to identify the actual sender and recipient of a message along with some other information such as date/time of transmission and intention of sender. It involves investigating metadata, port scanning as well as keyword searching.

Some of the common techniques which can be used for email forensic investigation are

- Header Analysis
- Server investigation
- Network Device Investigation
- Sender Mailer Fingerprints
- Software Embedded Identifiers

3. Algorithm:

A.Extraction of Information from EML files

EML files are basically emails in file format which are widely used for storing email messages. They are structured text files that are compatible across multiple email clients such as Microsoft Outlook, Outlook Express, and Windows Live Mail.

An EML file stores email headers, body content, attachment data as plain text. It uses base64 to encode binary data and Quoted-Printable (QP) encoding to store content information. The Python script that can be used to extract information from EML file is given below –

First, import the following Python libraries as shown below –

```
from __future__ import print_function
from argparse import ArgumentParser, FileType
from email import message_from_file

import os
import quopri
import base64
```

In the above libraries, quopri is used to decode the QP encoded values from EML files. Any base64 encoded data can be decoded with the help of the base64 library.

Next, let us provide an argument for the command-line handler. **Note that here it will accept only one argument which would be the path to EML file as shown below –**

```
if __name__ == '__main__':
   parser = ArgumentParser('Extracting information from EML file')
   parser.add_argument("EML_FILE",help="Path to EML File", type=FileType('r'))
   args = parser.parse_args()
   main(args.EML_FILE)
```

Now, we need to define the main() function in which we will use the method named message_from_file() from the email library to read the file-like object. Here we will access the headers, body content, attachments and other payload information by using resulting variable named emlfile as shown in the code given below –

```
def main(input_file):
   emlfile = message_from_file(input_file)
   for key, value in emlfile._headers:
      print("{}: {}".format(key, value))
print("\nBody\n")

if emlfile.is_multipart():
   for part in emlfile.get_payload():
      process_payload(part)
else:
   process_payload(emlfile[1])
```

Now, we need to define the process_payload() method in which we will extract message body content by using get_payload() method. We will decode QP encoded data by using the quopri.decodestring() function. We will also check the content MIME type so that it can handle the storage of the email properly. Observe the code given below −

```
def process_payload(payload):
   print(payload.get_content_type() + "\n" + "=" * len(payload.get_content_t
   body = quopri.decodestring(payload.get_payload())

   if payload.get_charset():
      body = body.decode(payload.get_charset())
else:
   try:
      body = body.decode()
   except UnicodeDecodeError:
      body = body.decode('cp1252')

if payload.get_content_type() == "text/html":
   outfile = os.path.basename(args.EML_FILE.name) + ".html"
   open(outfile, 'w').write(body)
elif payload.get_content_type().startswith('application'):
   outfile = open(payload.get_filename(), 'wb')
   body = base64.b64decode(payload.get_payload())
   outfile.write(body)
   outfile.close()
   print("Exported: {}\n".format(outfile.name))
else:
   print(body)
```

After executing the above script, we will get the header information along with various payloads on the console.

B. Structuring MBOX files from Google Takeout using Python

MBOX files are text files with special formatting that split messages stored within. They are often found in association with UNIX systems, Thunderbolt, and Google Takeouts.

In this section, you will see a Python script, where we will be structuring MBOX files got from Google Takeouts. But before that we must know how we can generate these MBOX files by using our Google or Gmail account.

i. Acquiring Google Account Mailbox into MBX Format

Acquiring Google account mailbox implies taking backup of our Gmail account. Backup can be taken for various personal or professional reasons. Note that Google provides backing up of Gmail data. To acquire our Google account mailbox into MBOX format, you need to follow the steps given below −

- Open My account dashboard.
- Go to the Personal info & privacy section and select Control your content link.
- You can create a new archive or can manage an existing one. If we click, CREATE ARCHIVE link, then we will get some checkboxes for each Google product we wish to include.
- After selecting the products, we will get the freedom to choose file type and maximum size for our archive along with the delivery method to select from the list.
- Finally, we will get this backup in MBOX format.

ii. Now, the MBOX file discussed above can be structured using Python as shown below −

First, need to import Python libraries as follows −

```
from __future__ import print_function
from argparse import ArgumentParser

import mailbox
import os
import time
import csv
from tqdm import tqdm

import base64
```

All the libraries have been used and explained in earlier scripts, except the mailbox library which is used to parse MBOX files.

iii. Now, provide an argument for the command-line handler. Here it will accept two arguments– one would be the path to the MBOX file, and the other would be the desired output folder.

```
if __name__ == '__main__':
    parser = ArgumentParser('Parsing MBOX files')
    parser.add_argument("MBOX", help="Path to mbox file")
    parser.add_argument(
        "OUTPUT_DIR",help = "Path to output directory to write report ""and exp
    args = parser.parse_args()
    main(args.MBOX, args.OUTPUT_DIR)
```

iv. Now, will define main() function and call mbox class of mailbox library with the help of which we can parse a MBOX file by providing its path –

```
def main(mbox_file, output_dir):
    print("Reading mbox file")
    mbox = mailbox.mbox(mbox_file, factory=custom_reader)
    print("{} messages to parse".format(len(mbox)))
```

v. Now, create some variables for further processing as follows –

```
parsed_data = []
attachments_dir = os.path.join(output_dir, "attachments")

if not os.path.exists(attachments_dir):
   os.makedirs(attachments_dir)
columns = [
   "Date", "From", "To", "Subject", "X-Gmail-Labels", "Return-Path", "Receive
   "Content-Type", "Message-ID","X-GM-THRID", "num_attachments_exported", "ex
```

vi. Next, use tqdm to generate a progress bar and to track the iteration process as follows –

```
for message in tqdm(mbox):
   msg_data = dict()
   header_data = dict(message._headers)
for hdr in columns:
   msg_data[hdr] = header_data.get(hdr, "N/A")
```

vii. Now, data needs to be appended. Then we will call create_report() method as follows –

```
parsed_data.append(msg_data)
create_report(
   parsed_data, os.path.join(output_dir, "mbox_report.csv"), columns)
def write_payload(msg, out_dir):
   pyld = msg.get_payload()
   export_path = []

if msg.is_multipart():
   for entry in pyld:
      export_path += write_payload(entry, out_dir)
else:
   content_type = msg.get_content_type()
   if "application/" in content_type.lower():
      content = base64.b64decode(msg.get_payload())
      export_path.append(export_content(msg, out_dir, content))
   elif "image/" in content_type.lower():
      content = base64.b64decode(msg.get_payload())
      export_path.append(export_content(msg, out_dir, content))
```

```
elif "video/" in content_type.lower():
   content = base64.b64decode(msg.get_payload())
   export_path.append(export_content(msg, out_dir, content))
elif "audio/" in content_type.lower():
   content = base64.b64decode(msg.get_payload())
   export_path.append(export_content(msg, out_dir, content))
elif "text/csv" in content_type.lower():
   content = base64.b64decode(msg.get_payload())
   export_path.append(export_content(msg, out_dir, content))
elif "info/" in content_type.lower():
   export_path.append(export_content(msg, out_dir,
   msg.get_payload()))
elif "text/calendar" in content_type.lower():
   export_path.append(export_content(msg, out_dir,
   msg.get_payload()))
elif "text/rtf" in content_type.lower():
   export_path.append(export_content(msg, out_dir,
   msg.get_payload()))
else:
   if "name=" in msg.get('Content-Disposition', "N/A"):
      content = base64.b64decode(msg.get_payload())
   export_path.append(export_content(msg, out_dir, content))
elif "name=" in msg.get('Content-Type', "N/A"):
   content = base64.b64decode(msg.get_payload())
   export_path.append(export_content(msg, out_dir, content))
eturn export_path
```

viii. Now, we need to define a method that will extract the filename from the msg object as follows –

```
def export_content(msg, out_dir, content_data):
   file_name = get_filename(msg)
   file_ext = "FILE"

   if "." in file_name: file_ext = file_name.rsplit(".", 1)[-1]
   file_name = "{}_{:.4f}.{}".format(file_name.rsplit(".", 1)[0], time.time(
   file_name = os.path.join(out_dir, file_name)
```

ix. Now, with the help of following lines of code, you can actually export the file –

```
if isinstance(content_data, str):
   open(file_name, 'w').write(content_data)
else:
   open(file_name, 'wb').write(content_data)
return file_name
```

x. Now, let us define a function to extract filenames from the message to accurately represent the names of these files as follows –

```
def get_filename(msg):
   if 'name=' in msg.get("Content-Disposition", "N/A"):
      fname_data = msg["Content-Disposition"].replace("\r\n", " ")
      fname = [x for x in fname_data.split("; ") if 'name=' in x]
      file_name = fname[0].split("=", 1)[-1]
   elif 'name=' in msg.get("Content-Type", "N/A"):
      fname_data = msg["Content-Type"].replace("\r\n", " ")
      fname = [x for x in fname_data.split("; ") if 'name=' in x]
      file_name = fname[0].split("=", 1)[-1]
   else:
      file_name = "NO_FILENAME"
   fchars = [x for x in file_name if x.isalnum() or x.isspace() or x == "."]
   return "".join(fchars)
```

xi. Now, we can write a CSV file by defining the create_report() function as follows –

```
def create_report(output_data, output_file, columns):
   with open(output_file, 'w', newline="") as outfile:
      csvfile = csv.DictWriter(outfile, columns)
      csvfile.writeheader()
      csvfile.writerows(output_data)
```

Conclusion: successfully implemented the program for track email using e–mail header information.

Review Questions:

Q1. What is email tracing and tracking in cyber security?

Q2. What are email forensics tools?

Q3.What is the important information of the header in an email?

Q4. what are the techniques Used in Email Forensic Investigation