

# Softcopy – Practical-1.1

## # Code

```
#!/usr/bin/env python3
# BEGINNING OF CODE
import re
import json
from email import message_from_string, message_from_file
from email.parser import Parser
from datetime import datetime
import socket
import ipaddress

class EmailHeaderAnalyzer:
    """
    A comprehensive email header analyzer for forensic investigation
    """

    def __init__(self, email_content):
        """
        Initialize the analyzer with email content

        Args:
            email_content: Raw email string or file object
        """
        if isinstance(email_content, str):
            self.email = message_from_string(email_content)
        else:
            self.email = message_from_file(email_content)

        self.analysis_results = {}

    def extract_basic_headers(self):
        """Extract basic email header information"""
        headers = {
            'From': self.email.get('From', 'Not Found'),
            'To': self.email.get('To', 'Not Found'),
```

```

        'Subject': self.email.get('Subject', 'Not Found'),
        'Date': self.email.get('Date', 'Not Found'),
        'Message-ID': self.email.get('Message-ID', 'Not Found'),
        'Return-Path': self.email.get('Return-Path', 'Not Found'),
        'Reply-To': self.email.get('Reply-To', 'Not Found'),
        'MIME-Version': self.email.get('MIME-Version', 'Not Found'),
        'Content-Type': self.email.get('Content-Type', 'Not Found')
    }

    self.analysis_results['basic_headers'] = headers
    return headers

def extract_received_headers(self):
    """
    Extract and parse all 'Received' headers to trace email path
    Critical for tracking email transmission route
    """
    received_headers = self.email.get_all('Received', [])
    parsed_received = []

    for idx, received in enumerate(received_headers):
        hop_info = {
            'hop_number': idx + 1,
            'raw_header': received,
            'timestamp': self._extract_timestamp(received),
            'from_server': self._extract_from_server(received),
            'by_server': self._extract_by_server(received),
            'ip_address': self._extract_ip_from_received(received)
        }
        parsed_received.append(hop_info)

    self.analysis_results['received_headers'] = parsed_received
    return parsed_received

def extract_originating_ip(self):
    """
    Extract the originating IP address (X-Originating-IP)
    This is crucial for tracing the actual sender location
    """
    originating_ip = self.email.get('X-Originating-IP', None)

```

```

        if originating_ip:
            # Clean up IP address (remove brackets if present)
            originating_ip = re.search(r'(\d+\.\d+\.\d+\.\d+)',
originating_ip)
            if originating_ip:
                originating_ip = originating_ip.group(1)

        # If X-Originating-IP not found, try to get from first Received
header
        if not originating_ip:
            received_headers = self.email.get_all('Received', [])
            if received_headers:
                originating_ip =
self._extract_ip_from_received(received_headers[-1])

        self.analysis_results['originating_ip'] = originating_ip
        return originating_ip

def analyze_authentication(self):
    """
    Analyze email authentication headers (SPF, DKIM, DMARC)
    Helps detect spoofing and verify email authenticity
    """
    auth_results = {
        'SPF': self.email.get('Received-SPF', 'Not Found'),
        'DKIM-Signature': self.email.get('DKIM-Signature', 'Not
Found'),
        'Authentication-Results': self.email.get('Authentication-
Results', 'Not Found'),
        'ARC-Authentication-Results': self.email.get('ARC-
Authentication-Results', 'Not Found')
    }

    # Determine if email passed authentication
    spf_pass = 'pass' in str(auth_results['SPF']).lower()
    dkim_pass = 'DKIM-Signature' in str(auth_results['DKIM-
Signature'])

    auth_results['spf_passed'] = spf_pass
    auth_results['dkim_present'] = dkim_pass
    auth_results['likely_spoofed'] = not (spf_pass or dkim_pass)

```

```

self.analysis_results['authentication'] = auth_results
return auth_results

def extract_message_id(self):
    """
    Extract and analyze Message-ID
    Useful for tracking email threads and identifying patterns
    """
    message_id = self.email.get('Message-ID', 'Not Found')

    # Extract domain from Message-ID
    domain = None
    if message_id != 'Not Found':
        domain_match = re.search(r'@([a-zA-Z0-9.-]+)', message_id)
        if domain_match:
            domain = domain_match.group(1)

    message_id_info = {
        'message_id': message_id,
        'domain': domain
    }

    self.analysis_results['message_id_info'] = message_id_info
    return message_id_info

def analyze_sender_info(self):
    """
    Detailed analysis of sender information
    Extracts email addresses and identifies potential spoofing
    """
    from_header = self.email.get('From', '')
    return_path = self.email.get('Return-Path', '')

    # Extract email addresses
    from_email = self._extract_email_address(from_header)
    return_email = self._extract_email_address(return_path)

    # Check for mismatch (potential spoofing indicator)

```

```
        mismatch = from_email != return_email if from_email and
return_email else False
```

```
    sender_info = {
        'from_header': from_header,
        'from_email': from_email,
        'return_path': return_path,
        'return_email': return_email,
        'address_mismatch': mismatch,
        'potential_spoofing': mismatch
    }
```

```
    self.analysis_results['sender_analysis'] = sender_info
    return sender_info
```

```
def get_ip_geolocation_info(self, ip_address):
    """
    Get basic information about an IP address
    Note: For production, integrate with geolocation APIs
    """
    try:
        ip_obj = ipaddress.ip_address(ip_address)
        ip_info = {
            'ip': ip_address,
            'is_private': ip_obj.is_private,
            'is_global': ip_obj.is_global,
            'is_loopback': ip_obj.is_loopback,
            'version': ip_obj.version
        }

        # Try reverse DNS lookup
        try:
            hostname = socket.gethostbyaddr(ip_address)[0]
            ip_info['hostname'] = hostname
        except:
            ip_info['hostname'] = 'Reverse DNS lookup failed'

        return ip_info
    except ValueError:
        return {'error': 'Invalid IP address'}
```

```

def detect_suspicious_patterns(self):
    """
    Detect common patterns associated with email crimes
    """
    suspicious_indicators = []

    # Check for authentication failures
    if self.analysis_results.get('authentication',
    {}).get('likely_spoofed'):
        suspicious_indicators.append('Email failed authentication
checks (SPF/DKIM)')

    # Check for sender/return-path mismatch
    if self.analysis_results.get('sender_analysis',
    {}).get('potential_spoofing'):
        suspicious_indicators.append('Mismatch between From and
Return-Path addresses')

    # Check for missing Message-ID
    if self.analysis_results.get('message_id_info',
    {}).get('message_id') == 'Not Found':
        suspicious_indicators.append('Missing Message-ID (unusual for
legitimate emails)')

    # Check for suspicious keywords in subject
    subject = self.email.get('Subject', '').lower()
    suspicious_keywords = ['urgent', 'verify account', 'suspended',
'confirm', 'prize', 'winner']
    found_keywords = [kw for kw in suspicious_keywords if kw in
subject]
    if found_keywords:
        suspicious_indicators.append(f'Suspicious keywords in
subject: {", ".join(found_keywords)}')

    self.analysis_results['suspicious_indicators'] =
suspicious_indicators
    return suspicious_indicators

def generate_forensic_report(self):
    """
    Generate a comprehensive forensic analysis report
    """

```

```

# Run all analysis methods
self.extract_basic_headers()
self.extract_received_headers()
self.extract_originating_ip()
self.analyze_authentication()
self.extract_message_id()
self.analyze_sender_info()
self.detect_suspicious_patterns()

# Analyze originating IP if available
orig_ip = self.analysis_results.get('originating_ip')
if orig_ip:
    self.analysis_results['ip_analysis'] =
self.get_ip_geolocation_info(orig_ip)

return self.analysis_results

def print_report(self):
    """Print a formatted forensic report"""
    report = self.generate_forensic_report()

    print("="*80)
    print(" EMAIL FORENSIC ANALYSIS REPORT ".center(80, "="))
    print("="*80)
    print(f"\nGenerated: {datetime.now().strftime('%Y-%m-%d %H:%M:
%S')}}\n")

# Basic Headers
print("\n" + "="*80)
print("1. BASIC HEADER INFORMATION")
print("="*80)
for key, value in report['basic_headers'].items():
    print(f"{key:20s}: {value}")

# Sender Analysis
print("\n" + "="*80)
print("2. SENDER ANALYSIS")
print("="*80)
sender = report.get('sender_analysis', {})
print(f"From Email          : {sender.get('from_email', 'N/A')}")

```

```

        print(f"Return Email      : {sender.get('return_email',
'N/A')}}")
        print(f"Address Mismatch  : {sender.get('address_mismatch',
False)}}")
        print(f"Potential Spoofing : {sender.get('potential_spoofing',
False)}}")

        # Authentication
        print("\n" + "="*80)
        print("3. AUTHENTICATION ANALYSIS")
        print("="*80)
        auth = report.get('authentication', {})
        print(f"SPF Passed          : {auth.get('spf_passed', False)}}")
        print(f"DKIM Present           : {auth.get('dkim_present', False)}}")
        print(f"Likely Spoofed      : {auth.get('likely_spoofed',
True)}}")

        # Message ID
        print("\n" + "="*80)
        print("4. MESSAGE ID ANALYSIS")
        print("="*80)
        msg_id = report.get('message_id_info', {})
        print(f"Message ID              : {msg_id.get('message_id', 'N/A')}}")
        print(f"Domain                  : {msg_id.get('domain', 'N/A')}}")

        # Originating IP
        print("\n" + "="*80)
        print("5. ORIGINATING IP INFORMATION")
        print("="*80)
        print(f"Originating IP          : {report.get('originating_ip', 'Not
Found')}}")

        if 'ip_analysis' in report:
            ip_info = report['ip_analysis']
            print(f"IP Version              : IPv{ip_info.get('version',
'N/A')}}")
            print(f"Is Private               : {ip_info.get('is_private',
'N/A')}}")
            print(f"Is Global                : {ip_info.get('is_global',
'N/A')}}")
            print(f"Hostname                : {ip_info.get('hostname',
'N/A')}}")

```



```

# Transmission Path
print("\n" + "="*80)
print("6. EMAIL TRANSMISSION PATH")
print("="*80)
received = report.get('received_headers', [])
if received:
    for hop in received:
        print(f"\nHop {hop['hop_number']}:")
        print(f"  From Server    : {hop.get('from_server',
'N/A')}}")
        print(f"  By Server       : {hop.get('by_server', 'N/A')}}")
        print(f"  IP Address      : {hop.get('ip_address',
'N/A')}}")
        print(f"  Timestamp       : {hop.get('timestamp', 'N/A')}}")
    else:
        print("No Received headers found")

# Suspicious Indicators
print("\n" + "="*80)
print("7. SUSPICIOUS INDICATORS")
print("="*80)
indicators = report.get('suspicious_indicators', [])
if indicators:
    for idx, indicator in enumerate(indicators, 1):
        print(f"{idx}. {indicator}")
else:
    print("No suspicious indicators detected")

print("\n" + "="*80)
print(" END OF REPORT ".center(80, "="))
print("="*80)

def export_json(self, filename='email_analysis.json'):
    """Export analysis results to JSON file"""
    report = self.generate_forensic_report()
    with open(filename, 'w') as f:
        json.dump(report, f, indent=4)
    print(f"\nAnalysis exported to {filename}")

# Helper methods

```

```

def _extract_timestamp(self, received_header):
    """Extract timestamp from Received header"""
    timestamp_match = re.search(r';\s*(.+)$', received_header)
    return timestamp_match.group(1).strip() if timestamp_match else
'Not Found'

def _extract_from_server(self, received_header):
    """Extract 'from' server information"""
    from_match = re.search(r'from\s+([\s\S]+)', received_header,
re.IGNORECASE)
    return from_match.group(1) if from_match else 'Not Found'

def _extract_by_server(self, received_header):
    """Extract 'by' server information"""
    by_match = re.search(r'by\s+([\s\S]+)', received_header,
re.IGNORECASE)
    return by_match.group(1) if by_match else 'Not Found'

def _extract_ip_from_received(self, received_header):
    """Extract IP address from Received header"""
    ip_match = re.search(r'\[(\d+\.\d+\.\d+\.\d+)\]',
received_header)
    return ip_match.group(1) if ip_match else 'Not Found'

def _extract_email_address(self, header_value):
    """Extract email address from header value"""
    email_match = re.search(r'[\w\.-]+@[\w\.-]+\.\w+', header_value)
    return email_match.group(0) if email_match else None

def main():
    """
    Main function to demonstrate email header analysis
    """
    print("Email Header Analysis Tool for Digital Forensics\n")
    print("Choose input method:")
    print("1. Paste raw email content")
    print("2. Load from file")
    print("3. Use sample email")

    choice = input("\nEnter choice (1-3): ").strip()

```

```

    if choice == '1':
        print("\nPaste the raw email (including headers). Press Ctrl+D
(Linux/MacOS) or Ctrl+Z (Windows) when done: ")
        import sys
        email_content = sys.stdin.read()

    elif choice == '2':
        filename = input("Enter email file path: ").strip()
        try:
            with open(filename, 'r') as f:
                email_content = f.read()
        except FileNotFoundError:
            print(f"Error: File '{filename}' not found")
            return

    else:
        # Sample email for demonstration
        email_content = """From: sender@example.com
To: recipient@example.com
Subject: Urgent Account Verification Required
Date: Mon, 9 Oct 2025 10:30:00 +0530
Message-ID: <12345.67890@mail.example.com>
Return-Path: different@suspicious.com
Received: from mail.example.com ([192.168.1.100]) by server.example.com
with SMTP; Mon, 9 Oct 2025 10:30:00 +0530
Received: from client.suspicious.com ([203.0.113.45]) by mail.example.com
with ESMTP; Mon, 9 Oct 2025 10:29:55 +0530
X-Originating-IP: [203.0.113.45]
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8

This is a sample email body for forensic analysis.
"""

        # Create analyzer instance and generate report
        analyzer = EmailHeaderAnalyzer(email_content)
        analyzer.print_report()

        # Ask if user wants to export to JSON
        export = input("\nExport analysis to JSON? (y/n): ").strip().lower()

```

```

        if export == 'y':
            filename = input("Enter filename (default: email_analysis.json):
").strip()
            if not filename:
                filename = 'email_analysis.json'
            analyzer.export_json(filename)

if __name__ == "__main__":
    main()
# END OF CODE

```

## # Output

```

[overnion - Codes (/run/media/overnion/persistence/Files/git/sppu-be-comp-content/Cybe
$ python3 Practical-1.py
Email Header Analysis Tool for Digital Forensics

Choose input method:
1. Paste raw email content
2. Load from file
3. Use sample email

Enter choice (1-3): 3
=====
===== EMAIL FORENSIC ANALYSIS REPORT =====
=====
Generated: 2025-10-09 20:19:22

=====
1. BASIC HEADER INFORMATION
=====
From           : sender@example.com
To             : recipient@example.com
Subject        : Urgent Account Verification Required
Date           : Mon, 9 Oct 2025 10:30:00 +0530
Message-ID     : <12345.67890@mail.example.com>
Return-Path    : different@suspicious.com
Reply-To       : Not Found
MIME-Version   : 1.0
Content-Type   : text/plain; charset=utf-8

```

```

=====
2. SENDER ANALYSIS
=====
From Email      : sender@example.com
Return Email    : different@suspicious.com
Address Mismatch : True
Potential Spoofing : True

=====
3. AUTHENTICATION ANALYSIS
=====
SPF Passed      : False
DKIM Present    : False
Likely Spoofed  : True

=====
4. MESSAGE ID ANALYSIS
=====
Message ID      : <12345.67890@mail.example.com>
Domain          : mail.example.com

=====
5. ORIGINATING IP INFORMATION
=====
Originating IP  : 203.0.113.45
IP Version      : IPv4
Is Private      : True
Is Global       : False
Hostname        : Reverse DNS lookup failed

```

```

=====
6. EMAIL TRANSMISSION PATH
=====

Hop 1:
  From Server   : mail.example.com
  By Server     : server.example.com
  IP Address    : 192.168.1.100
  Timestamp     : Mon, 9 Oct 2025 10:30:00 +0530

Hop 2:
  From Server   : client.suspicious.com
  By Server     : mail.example.com
  IP Address    : 203.0.113.45
  Timestamp     : Mon, 9 Oct 2025 10:29:55 +0530

=====
7. SUSPICIOUS INDICATORS
=====
1. Email failed authentication checks (SPF/DKIM)
2. Mismatch between From and Return-Path addresses
3. Suspicious keywords in subject: urgent

=====
===== END OF REPORT =====
=====

Export analysis to JSON? (y/n): y
Enter filename (default: email_analysis.json):

Analysis exported to email_analysis.json

```