A Modular Framework for Email Sender Trust Evaluation Using DNS, Reputation Intelligence, and Behavioral Heuristics

Nikola Skendrovic
Plummer Associates Inc. , Security Operations Team

Work: nskendrovic@plummer.com
Personal: nik.skendrovic@gmail.com

Date: September 20, 2025

Abstract

Email remains a dominant communication channel—and a persistent attack vector. Spoofing, phishing, and impersonation tactics continue to exploit weaknesses in sender authentication protocols and user trust. While standards such as SPF, DKIM, and DMARC are widely adopted, misconfigurations and deceptive tactics still allow malicious actors to bypass traditional filters. Commercial email gateways offer scoring and filtering mechanisms, but often lack transparency, adaptability, and educational value.

This paper presents a modular, Python-based framework for evaluating email sender trustworthiness through a combination of technical validation and behavioral analysis. The system parses raw .eml files to extract sender and routing metadata, sanitizes domain strings to prevent malformed lookups, validates SPF and MX records via DNS queries, and assesses domain age using local WHOIS lookups with fallback handling. It integrates IP reputation scoring via AbuseIPDB and applies heuristic checks to detect suspicious sender display names, hash-like identifiers, and high-risk domain patterns (e.g., uncommon TLDs or excessive length).

A composite scoring model begins at 100 and applies structured penalties for missing authentication records, short domain age, high abuse scores, and behavioral anomalies. Final trust scores are classified into Low, Medium, or High risk categories, enabling scalable and interpretable sender assessments. The framework is fully scriptable, transparent, and extensible—designed for integration into security workflows, educational environments, and custom tooling. It offers a practical alternative to proprietary email filtering systems, empowering analysts, researchers, and developers to understand and refine sender reputation logic with clarity and control.

Keywords: Email Security, SPF, DKIM, DMARC, DNS, WHOIS, AbuseIPDB, Sender Reputation, Phishing Detection, Threat Intelligence

1. Introduction

Email spoofing and impersonation attacks continue to pose significant risks to organizations and individuals. Despite widespread adoption of SPF, DKIM, and DMARC, many domains remain misconfigured or vulnerable to abuse. Commercial email gateways offer filtering and scoring mechanisms, yet often lack transparency, customization, or educational value for analysts and learners.

This research introduces a modular sender analysis framework that combines technical validation with behavioral heuristics to improve detection accuracy. The system:

- Parses .eml files to extract sender metadata and routing headers
- Sanitizes domain strings to prevent malformed lookups
- Validates SPF and MX records via DNS queries
- Assesses domain age using local WHOIS lookups with fallback handling
- Queries IP reputation databases (e.g., AbuseIPDB)
- Detects suspicious sender display names and domain patterns
- Computes a trust score based on both technical and contextual indicators
- Classifies risk levels as Low, Medium, or High based on score thresholds

The scoring model begins at 100 and applies deductions for missing SPF, high abuse scores, short domain age, absent MX records, and behavioral anomalies such as hash-like sender names or suspicious top-level domains. This hybrid approach enables more nuanced assessments, reducing false positives and highlighting deceptive senders that may otherwise appear legitimate.

Designed for integration into broader security workflows or educational environments, the framework is scriptable, transparent, and extensible—offering a practical tool for threat analysts, researchers, and students alike.

2. Related Work

Email authentication and sender reputation have been the subject of extensive research and industry development over the past two decades. Protocols such as SPF (Sender Policy Framework), DKIM (DomainKeys Identified Mail), and DMARC (Domain-based Message Authentication, Reporting, and Conformance) were introduced to combat spoofing and impersonation by verifying sender identity and message integrity. While these protocols are widely adopted, studies have shown that misconfiguration remains common, and enforcement is inconsistent across domains and mail servers. Many domains publish SPF records but fail to align them with actual sending infrastructure, and DKIM signatures are often missing or invalid due to poor key management or relay interference.

Academic studies have explored phishing detection using lexical analysis of domains, machine learning on header and body features, and trust evaluation frameworks. For example, Fette et al. (2007) introduced one of the first large-scale phishing detection studies using lexical and behavioral cues. Abu-Nimeh et al. (2007) compared multiple machine learning techniques for phishing detection, highlighting trade-offs in precision and recall. More recent work, such as Marchal et al. (2017), examined URL-based and domain age heuristics, emphasizing their value in catching malicious senders.

Tools like SpamAssassin pioneered heuristic scoring by analyzing message content, header anomalies, and sender metadata. However, its rule-based system can be opaque and difficult to customize, especially for non-expert users. Commercial platforms such as Proofpoint, Mimecast, and Barracuda offer enterprise-grade filtering and threat detection, but their scoring logic is proprietary and inaccessible to most users. These systems often rely on internal threat intelligence feeds and machine learning models that cannot be audited or adapted by the end user. As a result, analysts and educators lack visibility into how sender trust is determined, and small organizations are left without affordable, transparent alternatives.

In parallel, the rise of public threat intelligence APIs has enabled more granular reputation analysis. Services like AbuseIPDB, VirusTotal, IPQualityScore, and Cisco Talos provide real-time data on IP addresses, domains, and URLs associated with malicious activity. These platforms are invaluable for enrichment and correlation, but they require manual integration and often impose rate limits or licensing constraints. Moreover, they typically focus on individual indicators rather than holistic sender evaluation.

This framework builds on these foundations by combining multiple validation layers into a unified, transparent scoring engine. It leverages DNS queries to validate SPF and MX records, parses email headers to extract sender IPs, and integrates with AbuseIPDB to assess reputation. It also incorporates WHOIS data to evaluate domain age—a critical but often overlooked signal in phishing detection. Unlike monolithic or opaque systems, this framework is modular and scriptable, allowing users to inspect, modify, and extend each component. The scoring model is intentionally simple and interpretable, enabling analysts to understand why a sender is flagged and adjust thresholds based on context or risk tolerance.

By bridging the gap between protocol-level validation, threat intelligence, and practical scoring, this framework offers a middle ground between academic rigor and operational usability. It empowers users to build their own sender trust logic, experiment with new signals, and integrate reputation analysis into broader security workflows. In doing so, it contributes to a growing movement toward open, explainable, and customizable cybersecurity tooling.

3. System Architecture

3.0 Prerequisites

Before deploying or experimenting with the sender reputation analysis framework, users should ensure the following prerequisites are met. These requirements span software dependencies, API access, and basic system configuration.

3.0.1 Programming Environment

- **Python 3.8+** is required for compatibility with libraries used in DNS resolution, email parsing, and API integration.
- A modern code editor (e.g., VS Code, PyCharm) is recommended for development and debugging.

3.0.2 Required Python Libraries

The following packages must be installed via pip:

```
pip install dnspython
pip install requests
pip install python-whois
pip install pyspf

Optional but recommended:
pip install email-validator
pip install rich  # For enhanced terminal output
```

3.0.3 API Access

- **AbuseIPDB API Key**: Required to query IP reputation scores. Users must register at abuseipdb.com and obtain a free or paid API key.
- Additional APIs (e.g., VirusTotal, IPQualityScore) can be integrated with minimal changes to the modular design.

3.0.4 Network and DNS Access

- The system must have outbound internet access to perform DNS queries and API calls.
- DNS resolution is performed using dnspython, which queries public DNS servers.

3.0.5 Input Format

- The framework expects email files in .eml format, which preserve full headers and MIME structure.
- Emails can be sourced from local archives, mail clients, or exported from inboxes using IMAP or Gmail API.

3.0.6 Secure API Key Management

Within the code examples, placeholders such as:

```
headers = {"Key": "YOUR_API_KEY", "Accept": "application/json"}
```

are intentionally provided.

Best practices for handling API keys include:
Use environment variables (via os.environ) instead of hardcoding keys into scripts.

Avoid committing keys to version control by adding them to .gitignore and using secret management tools (e.g., Vault, AWS Secrets Manager).

Implement key rotation policies and restrict scope/permissions of API keys where possible.

Example:

```
import os

API_KEY = os.getenv("ABUSEIPDB_API_KEY")  # Load securely from environment
headers = {"Key": API_KEY, "Accept": "application/json"}
```

This approach ensures that sensitive credentials remain secure, portable, and compliant with security best practices.

3.0.7 Exception Handling for Network Calls

Network operations (DNS lookups, API requests, WHOIS queries) are prone to timeouts, rate limits, or transient failures. To ensure reliability, the framework should use structured exception handling with retry and fallback mechanisms.

Example for AbuseIPDB requests:

```
def query_abuseipdb(ip, api_key):
    url = f"https://api.abuseipdb.com/api/v2/check?ipAddress={ip}"
    headers = {"Key": api_key, "Accept": "application/json"}
    try:
        response = requests.get(url, headers=headers, timeout=5)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.Timeout:
        print(f"Timeout querying AbuseIPDB for {ip}")
        return None
    except requests.exceptions.RequestException as e:
        print(f"API error for {ip}: {e}")
        return None
```

This ensures the system fails gracefully under adverse conditions instead of halting analysis entirely.

3.1 Email Parsing

Using Python's email module, the system parses raw .eml files to extract:

- From: header (sender identity)
- Received: headers (routing path)

```
msg = BytesParser(policy=policy.default).parsebytes(raw_email)
sender = msg['From']
received_headers = msg.get_all('Received', [])
```

3.2 SPF Validation

SPF records are retrieved via DNS TXT queries. Presence of v=spf1 indicates basic domain hygiene.

```
answers = dns.resolver.resolve(domain, 'TXT')
if 'v=spf1' in rdata.to_text():
    return True
```

"The current implementation only verifies presence of an SPF record, not alignment of sending IPs. Full SPF validation with pyspf can be added in future work.

3.3 MX Record Check

MX records confirm that the domain is configured to send/receive email.

```
answers = dns.resolver.resolve(domain, 'MX')
return len(answers) > 0
```

3.4 IP Extraction

Sender IP is extracted from Received: headers using regex:

```
ip_pattern = r'\[(\d{1,3}(?:\.\d{1,3}){3})\]'
match = re.search(ip_pattern, h)
```

3.5 AbuseIPDB Reputation Query

The sender IP is queried against AbuseIPDB to retrieve an abuse confidence score (0–100).

```
response = requests.get(
    f"https://api.abuseipdb.com/api/v2/check?ipAddress={ip}",
    headers={"Key": "YOUR_API_KEY", "Accept": "application/json"})
```

3.6 WHOIS and Domain Age

Domain age is calculated from WHOIS creation date:

```
w = whois.whois(domain)
age_days = (datetime.now() - w.creation_date).days
```

3.7 Behavioral Heuristics

To detect deceptive senders that pass technical checks, the system applies heuristic filters:

Suspicious sender names: Hash-like identifiers, system alerts, or invoice-style labels

Risky domain patterns: Uncommon TLDs (e.g., .ru, .cn) or excessively long domain names

```
def is_suspicious_name(sender):
    return bool(re.search(r"(access log|#[A-Z0-9]{6,}|system
alert|invoice|payment)", sender, re.IGNORECASE))

def is_suspicious_domain(domain):
    return domain.endswith('.ru') or domain.endswith('.cn') or len(domain) >
30
```

3.8 Extensibility

The framework is designed to be modular and easily extensible, enabling new features with minimal effort. Enhancements include:

- **Real-time analysis**: Integration with Gmail API or mail server logs for continuous monitoring.
- **Custom output formats**: Export results in CSV, JSON, or SIEM-compatible formats.
- **Additional threat intelligence APIs**: VirusTotal, IPQualityScore, Cisco Talos, or others.
- **Visualization**: Enhanced terminal output with rich, or dashboards for trust score reporting.

4. Threat Model

To contextualize its applicability, the framework operates under a defined threat model that clarifies which attack types it can reasonably detect, and which remain outside its scope.

Covered Threats

- Domain spoofing: Detection of missing or misconfigured SPF and MX records exposes forged domains.
- Impersonation attempts: Suspicious display names, hash-like identifiers, and “system alert” labels are penalized.
- Use of newly registered domains: WHOIS lookups highlight domains with short lifespans, which are frequently leveraged in phishing.
- Risky TLDs and anomalous domain structures: Domains with uncommon extensions (e.g., .ru, .cn) or excessive length are flagged.
- IPs with poor reputation: Integration with AbuseIPDB surfaces addresses associated with abuse reports.

Out-of-Scope Threats

- Compromised legitimate accounts: Attacks launched from valid, long-established domains with strong reputation will likely evade detection.
- Content-level phishing: The framework does not analyze the body of the email, attachments, or embedded URLs.
- Social engineering and spear phishing: Targeted attacks using personalized sender identities or real compromised infrastructure fall outside its rule-based trust scoring.
- Encrypted or obfuscated metadata: Advanced adversaries may attempt to bypass header analysis using custom relays or privacy protections.

5. Trust Scoring Model

The sender trust score begins at 100 and applies deductions based on both technical and behavioral indicators. This hybrid approach improves detection accuracy by penalizing not only misconfigurations and threat intelligence signals, but also suspicious naming patterns and domain characteristics.

Factor	**Condition**	**Penalty**
SPF	Missing or invalid	−30
AbuseIPDB abuse confidence score	Score >50	−50
AbuseIPDB abuse confidence score	Score >20	−20
Domain Age	<30 days	−25
Domain Age	30–180 days	−10
MX Record	Missing	−25
Sender Name	Contains hashes, alerts, or system tags	−20
Domain Pattern	Suspicious TLD (.ru, .cn) or long name	−15

The final score is capped between 0 and 100. Risk levels are classified as:

- **Low Risk**: Score >80
- **Medium Risk**: Score 51–80
- **High Risk**: Score ≤50

This model balances strict technical validation with contextual awareness, reducing false positives while flagging deceptive senders that might otherwise appear legitimate.

6. Results and Interpretation

```
The following senders were analyzed using the framework:
```

Exhibit A: Suspicious Sender – sneezekey.ru

```
📨 Email Sender Analysis Report
Sender: "Access Log: #NYKDNJNWW" <rqgoq@sneezekey.ru>
Sender Domain: sneezekey.ru
Sender IP: 57.129.5.0
SPF Valid: True
AbuseIPDB abuse confidence score: 0
Domain Age: 108 days
MX Record Present: True
Trust Score: 55
Risk Level: Medium
```

This email superficially complies with technical standards—SPF validation passed, MX records are present, and the IP address has no known abuse reports. The domain is 108 days old, which avoids the “newly registered” penalty but still falls within the range of caution.

Despite these clean signals, the trust score is reduced to **55**, placing the sender in the **Medium Risk** category. This is due to behavioral heuristics: the display name includes a hash-like identifier (#NYKDNJNWW) and a system-style label (Access Log), both of which are common in phishing and alert-style scams. Additionally, the .ru top-level domain is flagged as suspicious due to its frequent association with spam and fraud campaigns.

This result demonstrates the value of combining protocol-level validation with contextual analysis. A sender that passes SPF and MX checks may still pose a risk if its naming conventions, domain patterns, or metadata resemble known phishing tactics. The framework’s hybrid scoring model successfully identifies this discrepancy, preventing a false sense of security based on technical compliance alone.

Exhibit B: Trusted Sender – comptia.org

📨 Email Sender Analysis Report

Sender: CompTIA <noreplies@comptia.org>

Sender Domain: comptia.org

```
Sender IP: 54.240.94.156
SPF Valid: True
AbuseIPDB abuse confidence score: 0
Domain Age: 10995 days
MX Record Present: True
Trust Score: 100
Risk Level: Low
```

This email originates from noreplies@comptia.org, a sender associated with the well-established technology certification organization CompTIA. The domain comptia.org passes SPF validation and has active MX records, confirming proper mail server configuration. The sender IP 54.240.94.156 returns an AbuseIPDB abuse confidence score of 0, indicating no known abuse reports. The domain age is 10,995 days—approximately 30 years—signaling long-term legitimacy and stability.

No behavioral red flags were detected. The display name is clean and professional, the domain uses a trusted .org top-level domain, and there are no suspicious lexical patterns or anomalies in the sender metadata. As a result, the trust score remains at **100**, placing the sender in the **Low Risk** category.

This outcome demonstrates the framework's ability to recognize and reward legitimate senders that meet both technical and behavioral criteria. It also validates the scoring model's balance—ensuring that trusted organizations are not penalized unnecessarily while maintaining vigilance against deceptive signals. The CompTIA result serves as a benchmark for what a fully compliant and trustworthy sender should look like within the system.

7. Use Cases

- **Security Operations**: Triage suspicious emails with transparent scoring
- **Education**: Teach email authentication and reputation analysis
- **Forensics**: Investigate sender legitimacy in incident response
- **Automation**: Integrate into pipelines for batch analysis

7.1 Security Operations

In modern security operations centers (SOCs), analysts are inundated with alerts, logs, and email-based threats. This framework provides a transparent and modular tool for triaging suspicious emails based on sender reputation. Unlike black-box filters, it offers a clear breakdown of trust factors—SPF validation, IP reputation, domain age, and MX configuration—allowing analysts to make informed decisions quickly.

By integrating this tool into SOC workflows, teams can:

- Prioritize investigation of high-risk senders
- Flag spoofed or misconfigured domains
- Enrich email metadata with reputation scores
- Reduce false positives by contextualizing sender behavior

Its interpretability makes it especially valuable in environments where accountability and auditability are critical.

7.2 Education and Training

Email infrastructure and authentication protocols are notoriously opaque to newcomers. This framework demystifies the process by exposing how headers, DNS records, and reputation databases interact to determine sender legitimacy.

Educators and trainers can use it to:

- Demonstrate real-world SPF, MX, and WHOIS queries
- Teach students how to parse and interpret email headers

- Explore threat intelligence APIs in a hands-on way
- Build exercises around scoring and risk classification

By turning abstract concepts into tangible code and output, the framework fosters deeper understanding of email security fundamentals.

7.3 Digital Forensics and Incident Response

During a security incident, determining whether an email is legitimate or malicious is often a time-sensitive task. This framework supports forensic analysis by extracting and validating sender metadata in a structured, repeatable way.

Incident responders can use it to:

- Trace the origin of suspicious messages
- Validate whether the sender domain is properly configured
- Cross-reference IP addresses with abuse databases
- Document findings with clear scoring and rationale

Its modular design allows responders to adapt it to different environments, whether analyzing archived .eml files or live inbox traffic.

7.4 Automation and Integration

The framework is built for extensibility and automation. It can be integrated into larger pipelines for batch processing, continuous monitoring, or enrichment of email logs.

Automation use cases include:

- Processing large volumes of .eml files from archives or exports
- Enriching SIEM or SOAR platforms with sender reputation scores
- Triggering alerts or actions based on trust thresholds
- Scheduling periodic scans of inboxes via Gmail API

Its Python foundation and API-driven architecture make it easy to plug into existing systems, whether for daily triage or long-term threat intelligence correlation.

8. Limitations and Future Work

While this framework provides a transparent and modular approach to sender reputation analysis, several limitations remain. First, the scoring model may over-rely on technical signals such as SPF validity, MX presence, and WHOIS-based domain age, which can be spoofed or misconfigured even by legitimate senders. Conversely, phishing emails that technically comply with these standards may evade detection if behavioral indicators are not incorporated. The current implementation does not yet analyze sender display names, domain lexical patterns, or header anomalies that often signal malicious intent. Additionally, WHOIS lookups are performed locally and may fail for privacy-protected or registrar-blocked domains, resulting in incomplete age data. AbuseIPDB scoring depends on external API availability and may not reflect real-time threat intelligence for low-volume or newly active IPs. Finally, the trust score is static and rule-based, lacking adaptive learning or contextual awareness. Future iterations may integrate machine learning, behavioral heuristics, and enriched metadata to improve accuracy and reduce false positives.

Limitations

- SPF validation does not enforce IP match
- DKIM and DMARC not yet implemented
- IPv6 not supported
- Reputation sources limited to AbuseIPDB

Future Enhancements

- Add DKIM and DMARC validation
- Support IPv6 and ASN lookup
- Integrate VirusTotal, IPQualityScore, Talos
- Build CLI or web dashboard
- Enable Gmail API integration for real-time analysis

8.1 Ethical and Legal Considerations

This framework is designed strictly for defensive and educational purposes, not for offensive use or reconnaissance. Its goal is to help analysts, researchers, and students understand how sender reputation can be evaluated transparently.
Key ethical and legal points include:

- **Responsible Use:** The framework must only be applied to email sources that the user is authorized to analyze (e.g., organizational inboxes, exported training datasets, or incident response evidence). Unauthorized scanning of third-party domains or addresses without consent may violate acceptable use policies or legal statutes.
- **Non-Malicious Intent:** The scoring logic and heuristics should never be misused for reconnaissance or aiding in spam/phishing campaigns. By explicitly framing the tool as an educational and defensive mechanism, the framework aligns with ethical research standards and supports responsible cybersecurity practices.
- **Transparency & Education:** Because the framework is modular and interpretable, it can serve as a teaching aid for ethical cybersecurity training programs, where students learn how to identify, classify, and defend against malicious senders.

9. Conclusion

This framework offers a transparent, extensible approach to email sender trust evaluation. By combining DNS validation, IP reputation, and domain metadata, it empowers users to assess risk with clarity and control. Whether used for education, automation, or security operations, this tool fills a critical gap between raw email headers and actionable insight.

Unlike proprietary filtering systems that often obscure their scoring logic, this framework prioritizes interpretability and modularity. Each component—from SPF enforcement to WHOIS-based domain age analysis—can be independently audited, tuned, or replaced. This makes the system ideal not only for operational use but also for research, training, and experimentation.

The framework's reliance on open standards and public APIs ensures accessibility for small teams, educators, and independent analysts. Its Python-based architecture allows rapid integration into existing workflows, whether for batch processing .eml files, enriching SIEM logs, or powering real-time inbox monitoring via Gmail API.

Furthermore, the scoring model provides a foundation for future enhancements. As threat intelligence evolves, additional signals—such as DKIM, DMARC, ASN reputation, geolocation, and behavioral heuristics—can be incorporated without disrupting the core logic. This adaptability positions the framework as a living tool that can grow alongside the threat landscape.

In an era where email remains both indispensable and vulnerable, empowering users with transparent, customizable sender analysis is not just a technical achievement—it's a strategic necessity. This framework bridges the gap between raw infrastructure and informed decision-making, offering a practical, ethical, and scalable solution for modern email security.

10. References

1. AbuseIPDB. (2025). AbuseIPDB API Documentation. Retrieved from https://www.abuseipdb.com
2. ICANN. (2025). WHOIS Protocol Overview. Retrieved from https://whois.icann.org
3. Levine, J., & Crocker, D. (2014). RFC 7208: Sender Policy Framework (SPF) for Authorizing Use of Domains in Email. Internet Engineering Task Force. Retrieved from https://tools.ietf.org/html/rfc7208
4. Mockapetris, P. (1987). RFC 1035: Domain Names—Implementation and Specification. Internet Engineering Task Force. Retrieved from https://tools.ietf.org/html/rfc1035
5. Python Software Foundation. (2025). Python 3 Standard Library Documentation. Retrieved from https://docs.python.org/3
6. I. Fette, N. Sadeh, and A. Tomasic, "Learning to detect phishing emails," *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*, Banff, AB, Canada, pp. 649–656, ACM, 2007. doi: 10.1145/1242572.1242660.
7. S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," *Proceedings of the Anti-Phishing Working Groups 2nd Annual eCrime Researchers Summit (eCrime '07)*, Pittsburgh, PA, USA, pp. 60–69, IEEE, 2007. doi: 10.1109/ECRIME.2007.4408060.
8. S. Marchal, N. Asankan, J. François, R. State, and T. Engel, "PhishStorm: Detecting phishing with streaming analytics," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 58–71, 2017. doi: 10.1109/TNSM.2016.2647203.

Appendix A: Full Source Code (With Behavioral detection)

```
# Import necessary modules for email parsing, DNS queries, HTTP requests, regex,
WHOIS lookup, and date handling
import email
from email import policy
from email.parser import BytesParser
import dns.resolver
import requests
import re
import whois
from datetime import datetime

# Parse the raw email and extract sender and routing headers
def parse_email(raw_email):
    # Parse the email using default policy (handles headers and encoding cleanly)
    msg = BytesParser(policy=policy.default).parsebytes(raw_email)
    # Extract the 'From' header (sender's email address)
    sender = msg['From']
    # Extract all 'Received' headers (used to trace sender IP)
    received_headers = msg.get_all('Received', [])
    return sender, received_headers

# Check if the domain has a valid SPF record
def check_spf(domain):
    try:
        # Query DNS TXT records for the domain
        answers = dns.resolver.resolve(domain, 'TXT')
        for rdata in answers:
            # Look for an SPF record starting with 'v=spf1'
            if 'v=spf1' in rdata.to_text():
                return True
    except Exception:
        # If DNS query fails or no SPF found
        return False
    return False

# Extract the sender IP address from the Received headers
def extract_ip(headers):
    # Regex pattern to match IPv4 addresses in square brackets
    ip_pattern = r'\[(\d{1,3}(?:\.\d{1,3}){3})\]'
    for h in headers:
        match = re.search(ip_pattern, h)
        if match:
            return match.group(1)
    return None

# Use WHOIS to determine domain age in days
def get_domain_age_local(domain):
    try:
        w = whois.whois(domain)
        creation_date = w.creation_date

        # Handle list of dates
        if isinstance(creation_date, list):
            creation_date = next((d for d in creation_date if isinstance(d,
datetime)), None)

        # Final check
        if not isinstance(creation_date, datetime):
            print(f"WHOIS failed for {domain}: no valid creation date")
            return -1

        age_days = (datetime.now() - creation_date).days
        return age_days
    except Exception as e:
        print(f"Local WHOIS error for {domain}: {e}")
        return -1

# Check if the domain has MX records (mail server configuration)
def has_mx_record(domain):
    try:
        answers = dns.resolver.resolve(domain, 'MX')
        return len(answers) > 0
    except Exception:
        return False

# Query AbuseIPDB for sender IP reputation
def abuseipdb_check(ip):
    try:
        response = requests.get(
            f"https://api.abuseipdb.com/api/v2/check?ipAddress={ip}",
            headers={
                "Key": "YOUR_API_KEY",  # Replace with your actual AbuseIPDB API
key
                "Accept": "application/json"
            }
        )
        data = response.json()
```

```python
        # Extract abuse confidence score (0-100)
        return data.get('data', {}).get('abuseConfidenceScore', 0)
    except Exception:
        # Return 0 if API call fails
        return 0

# Calculate a trust score based on multiple factors
def score_sender(spf_valid, abuse_score, domain_age, mx_valid, sender, domain):
    score = 100  # Start with full trust

    # Technical signals
    if not spf_valid:
        score -= 30
    if abuse_score > 50:
        score -= 50
    elif abuse_score > 20:
        score -= 20
    if domain_age != -1:
        if domain_age < 30:
            score -= 25
        elif domain_age < 180:
            score -= 10
    if not mx_valid:
        score -= 25

    # Behavioral signals
    if is_suspicious_name(sender):
        score -= 20
    if is_suspicious_domain(domain):
        score -= 15

    # Cap score between 0 and 100
    score = max(0, min(score, 100))
    return score

# Load and analyze a local .eml email file
with open(r'C:\01. Python TEST\test2.eml', 'rb') as f:
    raw_email = f.read()

# Detect suspicious sender display names
def is_suspicious_name(sender):
    return bool(re.search(r"(access log|#[A-Z0-9]{6,}|system
alert|invoice|payment)", sender, re.IGNORECASE))

# Detect suspicious domains or TLDs
```

```python
def is_suspicious_domain(domain):
    return domain.endswith('.ru') or domain.endswith('.cn') or len(domain) > 30

# Run analysis pipeline
sender, headers = parse_email(raw_email)
domain = sender.split('@')[-1]  # Extract domain from sender email
domain = re.sub(r"[^\w\.-]", "", domain.strip().lower())
domain_age = get_domain_age_local(domain)
mx_valid = has_mx_record(domain)
spf_valid = check_spf(domain)
ip = extract_ip(headers)
abuse_score = abuseipdb_check(ip)
trust_score = score_sender(spf_valid, abuse_score, domain_age, mx_valid, sender,
domain)

# Print full sender reputation report
print("📧 Email Sender Analysis Report")
print(f"Sender: {sender}")
print(f"Sender Domain: {domain}")
print(f"Sender IP: {ip}")
print(f"SPF Valid: {spf_valid}")
print(f"AbuseIPDB abuse confidence score: {abuse_score}")
print(f"Domain Age: {domain_age} days")
print(f"MX Record Present: {mx_valid}")
print(f"Trust Score: {trust_score}")
print("Risk Level:", "Low" if trust_score > 80 else "Medium" if trust_score > 50
else "High")
```