Email Security Tool: Code Explanation and Analysis

Technical Documentation

March 10, 2025

Contents

1 Introduction

This document provides a detailed technical explanation of the Python application for email security analysis. The tool is designed to help users detect potentially malicious emails by analyzing their content for signs of spam and phishing attacks. It combines machine learning techniques with rule-based heuristics to provide comprehensive security analysis.

2 Application Overview

Key Features

The application provides three main functionalities:

- **Spam Detection**: Identifying unsolicited bulk email
- **Phishing Detection**: Detecting fraudulent attempts to steal sensitive information
- **Model Training**: Training custom detection models with user-provided datasets

The code is structured as a single class `EmailSecurityApp` that manages the application's functionality through various methods. The application uses a tabbed interface to separate its main functions, making it more user-friendly and intuitive.

3 Technical Architecture

3.1 Dependencies

The application relies on several Python libraries to function:

- `tkinter`: GUI framework for creating the application interface
- `pandas`: Data manipulation library for handling datasets
- `scikit-learn`: Machine learning library providing:
  - `TfidfVectorizer` for text feature extraction
  - `MultinomialNB` for Naive Bayes classification
  - Training and evaluation functions
- `re`: Regular expression library for pattern matching
- `pickle`: Object serialization for saving/loading models
- `threading`: For background processing
- Additional libraries for URL handling, email parsing, etc.

3.2 Class Structure

The application is organized around a single main class with numerous methods:

```
class EmailSecurityApp:
    def __init__(self, root)  # Initialize the application
    def setup_example_datasets()  # Create sample datasets
    def create_tabs()  # Set up the tabbed interface
    def setup_spam_tab()  # Configure spam detection UI
    def setup_phishing_tab()  # Configure phishing detection UI
    def setup_training_tab()  # Configure model training UI
```

```
    def toggle_dataset_entry()  # Handle dataset selection UI
    def load_email_file()  # Load email content from file
    def browse_dataset()  # Select custom dataset files
    def update_text_widget()  # Helper for updating text displays
    def load_models()  # Load pre-trained models if available
    def check_spam()  # Perform spam detection analysis
    def check_phishing()  # Perform phishing detection analysis
    def train_model()  # Train ML models on datasets
    def _train_model_thread()  # Background thread for model training
```

4 Machine Learning Implementation

4.1 Text Feature Extraction

The application uses the TF-IDF (Term Frequency-Inverse Document Frequency) technique to convert email text into numerical features suitable for machine learning:

```
self.vectorizer = TfidfVectorizer(max_features=5000)
X_train_features = self.vectorizer.fit_transform(X_train)
```

This technique:

- Converts text into numerical vectors
- Gives higher weight to important terms
- Reduces the impact of common words
- Limits features to 5000 to prevent overfitting

4.2 Classification Model

The application uses the Multinomial Naive Bayes classifier for both spam and phishing detection:

```
self.spam_model = MultinomialNB()
self.spam_model.fit(X_train_features, y_train)
```

Multinomial Naive Bayes is chosen because:

- It works well with text classification problems
- It's efficient with high-dimensional data
- It can provide probability estimates for predictions
- It performs well even with relatively small training datasets

4.3 Model Training Process

The training process follows these steps:

1. Load and prepare the dataset (built-in or custom)
2. Split data into training and testing sets (80%/20%)
3. Create and fit the TF-IDF vectorizer
4. Train the Naive Bayes classifier
5. Evaluate model performance
6. Save the trained model to disk

5 Detection Techniques

5.1 Spam Detection

The spam detection functionality combines machine learning with rule-based checks:

```
# Machine learning classification
features = self.vectorizer.transform([email_content])
prediction = self.spam_model.predict(features)[0]
probability = self.spam_model.predict_proba(features)[0]

# Rule-based checks
spam_phrases = ["viagra", "lottery", "winner", ...]
for phrase in spam_phrases:
    if phrase in email_content.lower():
        spam_indicators.append(f"Contains spam phrase: '{phrase}'")

# Check for excessive capital letters and punctuation
capitals_ratio = sum(1 for c in email_content if c.isupper()) / max(len(email_content), 1)
if capitals_ratio > 0.3:
    spam_indicators.append(f"Excessive capital letters ({capitals_ratio*100:.1f}%)")
```

5.2 Phishing Detection

The phishing detection is more comprehensive, using multiple techniques:

5.2.1 URL Analysis

```
# Extract URLs
url_pattern = r'https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+[/\w\.-]*'
urls = re.findall(url_pattern, email_content)

# Analyze URL characteristics
for url in urls:
    parsed_url = urllib.parse.urlparse(url)
    if any(c in parsed_url.netloc for c in ['@', '..']):
        result += "         Suspicious URL format\n"
        phishing_score += 1
```

5.2.2 Form Detection

```
# Extract potential login forms
form_pattern = r'<form.*?>.*?</form>'
forms = re.findall(form_pattern, email_content, re.DOTALL | re.IGNORECASE)
```

5.2.3 Content Analysis

```
# Check for password or credential requests
credential_patterns = [
    r'password', r'login', r'sign in', r'verify your account',
    r'update your information', r'credit card', r'ssn', r'social security'
]

# Check for sense of urgency
urgency_patterns = [
    r'urgent', r'immediate action', r'within 24 hours', r'account.*?suspend',
    r'limited time', r'act now', r'immediately'
]
```

5.2.4 Domain Verification

```
# Try to verify if domain exists
try:
    socket.gethostbyname(parsed_url.netloc)
except:
    spoof_indicators.append(f"Domain does not exist: {parsed_url.netloc}")
    phishing_score += 3
```

5.2.5 Risk Scoring

The application calculates a phishing risk score based on all indicators and classifies emails as:

- **High Risk** (score $> 70\%$): Very likely to be a phishing attempt
- **Moderate Risk** (score $> 40\%$): Contains suspicious elements
- **Low Risk** (score $\leq 40\%$): Likely legitimate

6 User Interface Implementation

6.1 Tabbed Interface

The application uses ttk.Notebook to create a tabbed interface:

```
self.notebook = ttk.Notebook(self.root)
self.notebook.pack(fill=tk.BOTH, expand=1, padx=10, pady=10)

# Create tabs
self.tab_spam = tk.Frame(self.notebook, bg="#f0f0f0")
self.tab_phishing = tk.Frame(self.notebook, bg="#f0f0f0")
self.tab_training = tk.Frame(self.notebook, bg="#f0f0f0")

self.notebook.add(self.tab_spam, text="Spam Detection")
self.notebook.add(self.tab_phishing, text="Phishing Detection")
self.notebook.add(self.tab_training, text="Model Training")
```

6.2 Input and Results Display

Each functionality tab contains:

- Text input area for email content
- Control buttons for operations
- Scrollable text area for displaying results

6.3 Email File Loading

The application supports loading emails from .eml files:

```
if file_path.endswith('.eml'):
    with open(file_path, 'rb') as fp:
        msg = BytesParser(policy=policy.default).parse(fp)
        if msg.get_content_type() == 'text/plain':
            text = msg.get_content()
        else:
            # Handle multipart messages
            text = ""
            for part in msg.iter_parts():
                if part.get_content_type() == 'text/plain':
                    text += part.get_content()
```

7 Example Datasets

The application includes built-in datasets for both spam and phishing detection:

```
spam_data = {
    'text': [
        "Congratulations! You've won a million dollars in our lottery!",
        "Please verify your account details by clicking this link",
        # ... more examples ...
        "Could you please review the attached document before Friday?",
    ],
    'label': [
        1, 1, 1, 1, 1, 1, 1,  # Spam examples
        0, 0, 0, 0, 0, 0, 0, 0  # Non-spam examples
    ]
}
```

```
phishing_data = {
    'text': [
        "Dear customer, your bank account has been locked. Click http://fake-bank.com to
    verify your identity.",
        # ... more examples ...
        "Your subscription to The New York Times has been confirmed."
    ],
    'is_phishing': [
        1, 1, 1, 1, 1, 1, 1,  # Phishing examples
        0, 0, 0, 0, 0, 0, 0, 0  # Legitimate examples
    ]
}
```

8 Multi-threading Implementation

The application uses threading to prevent the UI from freezing during model training:

```
# Start training in a separate thread
threading.Thread(target=self._train_model_thread, args=(model_type, dataset_path)).start()
```

This allows the application to remain responsive while performing computationally intensive tasks.

9 Model Persistence

Trained models are saved to disk using pickle, allowing them to be reused across sessions:

```
# Save models
with open(model_filename, 'wb') as f:
    pickle.dump(model_to_save, f)

if not os.path.exists('vectorizer.pkl'):
    with open('vectorizer.pkl', 'wb') as f:
        pickle.dump(self.vectorizer, f)
```

10 Security Considerations

The tool implements several layers of security analysis:

- **Content analysis** for detecting suspicious language patterns
- **URL inspection** for identifying potentially malicious links
- **Form detection** for identifying credential harvesting attempts
- **Domain verification** to check if linked domains actually exist
- **Combined ML and rule-based approach** for better detection accuracy

11 Conclusion

The Email Security Tool demonstrates an effective approach to email security through a combination of machine learning and traditional rule-based analysis. The application provides:

- Comprehensive detection of spam and phishing threats
- User-friendly interface accessible to non-technical users
- Flexibility to use built-in or custom datasets
- Multiple layers of security analysis
- Detailed explanations of detected threats

This application could be extended in the future with:

- Integration with email clients
- More sophisticated ML models (e.g., deep learning)
- Additional security checks
- Real-time threat intelligence integration
- Improved visualization of results