Security Technical Operational Report

# Technical Security Assessment Report

## Technical Summary This report details the findings of a technical security assessment conducted on the "DataMapping-on-Image-using-Python" application, a SaaS tool designed to generate PDF documents from structured data. The assessment involved automated static code analysis (SAST) and a software composition analysis (SCA) of its dependencies.

The automated scans of the repository, completed on 2025-10-13, \*\*did not identify any specific code-level vulnerabilities or known exploits within the declared project dependencies\*\*. The application's codebase is focused on data processing with Pandas and PDF generation with FPDF2, and the current implementation does not trigger any high-risk alerts from our "basic" scan tooling.

However, a manual review of the application's architecture, based on the provided README and project structure, highlights several areas for proactive security hardening. These recommendations focus on input validation, resource management, and secure file handling to mitigate potential risks such as Denial of Service (DoS) and data exposure, which are critical for any application operating in a SaaS environment. The following report provides detailed analysis and actionable recommendations to enhance the application's overall security posture.

## Vulnerability Analysis

### Code Security Issues The Static Application Security Testing (SAST) scan of the Python source code did not detect any common vulnerabilities. The analysis looked for issues such as command injection, insecure file I/O, hardcoded secrets, and cross-site scripting (XSS) within the Streamlit framework, with no findings reported. The code's primary function of mapping data coordinates to a PDF template appears to be implemented in a direct and secure manner. The absence of findings indicates good basic coding hygiene but does not preclude the existence of logic-based or architectural flaws.

### Dependency Vulnerabilities The Software Composition Analysis (SCA) performed on the `requirements.txt` file found no known published vulnerabilities for the specified versions of the dependencies: - `streamlit==1.34.0` - `pandas==2.2.2` - `fpdf2==2.7.7` - `openpyxl==3.1.2` - `Pillow==10.3.0`

While the current dependency tree is clean, it is crucial to implement a continuous monitoring strategy, as new vulnerabilities for these packages may be discovered in the future.

### Architecture Security Review The application's architecture, while simple, presents potential operational risks if deployed in a multi-tenant or publicly accessible SaaS environment. The primary areas of concern are:

1. \*\*Unvalidated Input Processing\*\*: The application directly processes data from user-uploaded Excel files. There is no indication of validation or sanitization on the data being read. Maliciously crafted input, such as excessively long strings in cells, could lead to unexpected behavior, broken PDF layouts, or high resource consumption during the rendering process.  
2. \*\*Resource Exhaustion Risk\*\*: The application supports bulk generation and zipping of PDF files. A user uploading an Excel file with a very large number of rows could trigger a process that consumes excessive memory and CPU, potentially leading to a Denial of Service (DoS) condition that affects the application server and all its users.  
3. \*\*Insecure File Uploads\*\*: The application assumes the uploaded file is a well-formed Excel sheet. It does not appear to validate the file type beyond its extension. This could allow users to upload malicious or unexpected file types, which could be a vector for further attacks if the server has other misconfigurations.  
4. \*\*Temporary File Management\*\*: The generation of multiple PDFs and a final `.zip` file involves creating temporary files on the server's filesystem. Without a robust cleanup mechanism, these files, which may contain sensitive receipt or invoice data, could be left behind, leading to data leakage or excessive disk space consumption.

## Detailed Technical Findings

### Critical Vulnerabilities No critical vulnerabilities were identified during this assessment.

### High-Risk Issues No high-risk vulnerabilities were identified during this assessment.

### Medium-Risk Issues No medium-risk vulnerabilities were identified during this assessment.

## Security Testing Results The following table summarizes the results from the automated security scans.

| Test Type | Description | Findings |  
| :--- | :--- | :--- |  
| \*\*Static Application Security Testing (SAST)\*\* | Analysis of the Python source code for security flaws. | \*\*0 Findings\*\* |  
| \*\*Software Composition Analysis (SCA)\*\* | Scan of project dependencies for known vulnerabilities. | \*\*0 Findings\*\* |  
| \*\*Total Issues\*\* | | \*\*0\*\* |

## Technical Recommendations Based on the architectural review, we recommend the following proactive security hardening measures:

1. \*\*Implement Strict Input Validation and Sanitization\*\*:  
 \* \*\*Description\*\*: Before processing data from the uploaded Excel file, validate its content. This includes checking the length of strings, data types, and potential malicious characters.  
 \* \*\*Risk Mitigation\*\*: Prevents layout corruption in generated PDFs and mitigates risks associated with rendering untrusted data, including potential DoS from oversized inputs.

2. \*\*Enforce Resource Limits on File Uploads and Processing\*\*:  
 \* \*\*Description\*\*: Introduce limits on the maximum allowable size of uploaded files and the maximum number of rows that can be processed in a single bulk operation.  
 \* \*\*Risk Mitigation\*\*: Protects the application server from resource exhaustion attacks (CPU, memory) and ensures service availability for all users.

3. \*\*Strengthen File Upload Security\*\*:  
 \* \*\*Description\*\*: Validate uploaded files based on their content (e.g., magic bytes) in addition to the file extension to ensure they are legitimate Excel files.  
 \* \*\*Risk Mitigation\*\*: Prevents users from uploading potentially malicious or irrelevant file types, reducing the server's attack surface.

4. \*\*Implement Secure Temporary File Management\*\*:  
 \* \*\*Description\*\*: Use a secure and reliable method for handling temporary files. Ensure that all generated PDFs and the final zip archive are created in a non-web-accessible directory and are purged immediately after the user downloads them or their session ends.  
 \* \*\*Risk Mitigation\*\*: Prevents sensitive data leakage from orphaned temporary files and ensures efficient use of disk space.

5. \*\*Establish a Dependency Management Strategy\*\*:  
 \* \*\*Description\*\*: Pin all direct and transitive dependencies using a lock file (e.g., `requirements.lock`). Integrate an automated tool like Dependabot to continuously monitor for new vulnerabilities and create pull requests for updates.  
 \* \*\*Risk Mitigation\*\*: Ensures a repeatable build environment and provides timely alerts for newly discovered vulnerabilities in the dependency chain.

## Implementation Guidelines The following are practical steps to implement the recommendations:

1. \*\*Input Validation\*\*:  
 \* Before passing data to `fpdf2`, iterate through the pandas DataFrame and check the length of string-based columns. For example: `if len(str(row['ClientName'])) > 100: handle\_error()`.  
 \* Consider using a library like `python-magic` to verify the MIME type of the uploaded file before it is processed by `pandas.read\_excel`.

2. \*\*Resource Limiting\*\*:  
 \* In the Streamlit file upload widget, check the size of the uploaded object: `if uploaded\_file.size > 10\_000\_000: st.error("File size cannot exceed 10MB.")`.  
 \* After loading the data, check the DataFrame's size: `if len(df) > 5000: st.error("Cannot process more than 5000 rows at a time.")`.

3. \*\*Secure Temporary File Handling\*\*:  
 \* Utilize Python's built-in `tempfile` module to create a temporary directory for each user session or request: `with tempfile.TemporaryDirectory() as temp\_dir: ...`.  
 \* Write all generated PDFs and the final `.zip` file into this `temp\_dir`. The `with` statement ensures the directory and its contents are automatically and reliably deleted afterward.

## Security Monitoring and Alerting To maintain security visibility for this application in a production environment, we recommend implementing the following monitoring and alerting capabilities:

1. \*\*Logging\*\*:  
 \* \*\*Activity Logging\*\*: Log key events such as file uploads (including filename, size, and user identifier if applicable) and bulk generation requests.  
 \* \*\*Error Logging\*\*: Implement structured logging for application errors. Specifically, log any failures during file validation, data processing, or PDF generation. High error rates could indicate probing or a malicious attempt to crash the service.

2. \*\*Alerting\*\*:  
 \* \*\*Resource Threshold Alerts\*\*: Configure alerts in your hosting environment to trigger if the application's CPU or memory usage exceeds predefined thresholds for a sustained period. This can serve as an early warning for a resource exhaustion attack.  
 \* \*\*Anomalous Activity Alerts\*\*: Set up alerts for an unusually high number of file uploads or generation requests from a single source IP address within a short time frame.  
 \* \*\*Critical Error Alerts\*\*: Configure alerts for critical exceptions, such as failures to clean up temporary file directories, which could indicate a potential data exposure risk.