NInet Security Evaluation Report

Manyfold

V 1.0
Amsterdam, June 8th, 2024
Public

Document Properties

Client	Manyfold
Title	NLnet Security Evaluation Report
Targets	Manyfold (https://github.com/manyfold3d/manyfold) Analysis of manyfold.app infrastructure
Version	1.0
Pentester	Stefan Vink
Authors	Stefan Vink, Abhinav Mishra
Reviewed by	Abhinav Mishra
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	May 30th, 2024	Stefan Vink	Initial draft
0.2	June 6th, 2024	Stefan Vink	Ready-for-Review
1.0	June 8th, 2024	Abhinav Mishra	Final Report

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between May 20, 2024 and May 30, 2024, Radically Open Security B.V. carried out a penetration test for Manyfold and NLnet NGI Zero Entrust.

This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the penetration test was limited to the following target(s):

- Manyfold (https://github.com/manyfold3d/manyfold)
- Analysis of manyfold.app infrastructure

The scoped services are broken down as follows:

- Pentesting: 4.25 days
- Reporting: 0.75 days
- **Total effort: 5 days**

1.3 Project objectives

ROS will perform a penetration test of the Manyfold Web Application with Manyfold, in order to assess the security posture. To do so, ROS will assess these and guide Manyfold in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The Security Audit took place between May 20, 2024 and May 30, 2024.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 High, 2 Elevated, 5 Moderate and 3 Low-severity issues.

The high-severity findings were due to insecure processing of compressed files (Zip Slip) and lack of file size/type validation which could lead to code execution, overwriting critical files, and denial of service. Additionally, insecure

session management was identified, with no session revocation on logout and no expiration, making sessions susceptible to hijacking and unauthorized access.

We also identified several elevated severity issues. The application lacked protection against zip decompression bombs, allowing large file uploads that could exhaust server resources and lead to denial of service. Moreover, the lack of password brute force protection during login increased the risk of account compromise. Another significant issue was the missing or insecure security headers, exposing the application to XSS attacks, clickjacking, and other security risks.

In addition to the high and elevated severity issues, we found numerous moderate and low severity issues that, while less critical individually, collectively pose significant risks. These included user enumeration through password and login forms, enabling attackers to enumerate valid usernames and email addresses, facilitating targeted attacks such as phishing or brute force. The application also includes external scripts from third parties, which could introduce malicious code if compromised.

Several misconfigurations were discovered, such as insecure Docker configurations and Docker containers running as root, increasing the risk of privilege escalation and persistent vulnerabilities. The library path misconfiguration allowed administrators to configure root directories as the path, significantly increasing the risk of unauthorized access and system compromises.

By exploiting these issues, an attacker might gain unauthorized access, steal sensitive data and disrupt services. Immediate remediation of high and elevated severity issues is crucial, followed by addressing the moderate and low severity issues to enhance the overall security posture.

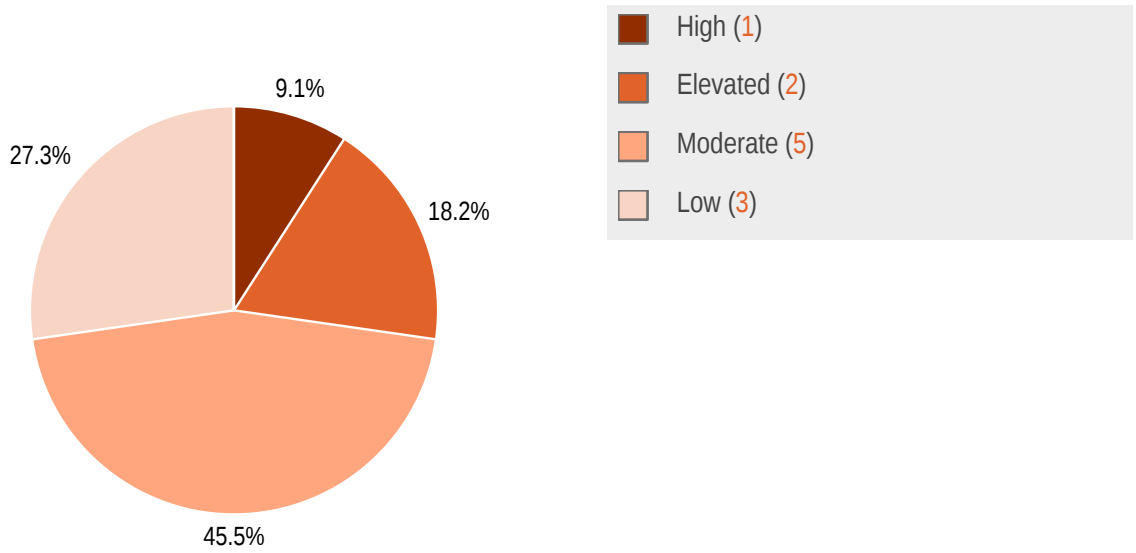
No issues were found in the supporting infrastructure that are used by the manyfold.app and it's subdomains.

1.6 Summary of Findings

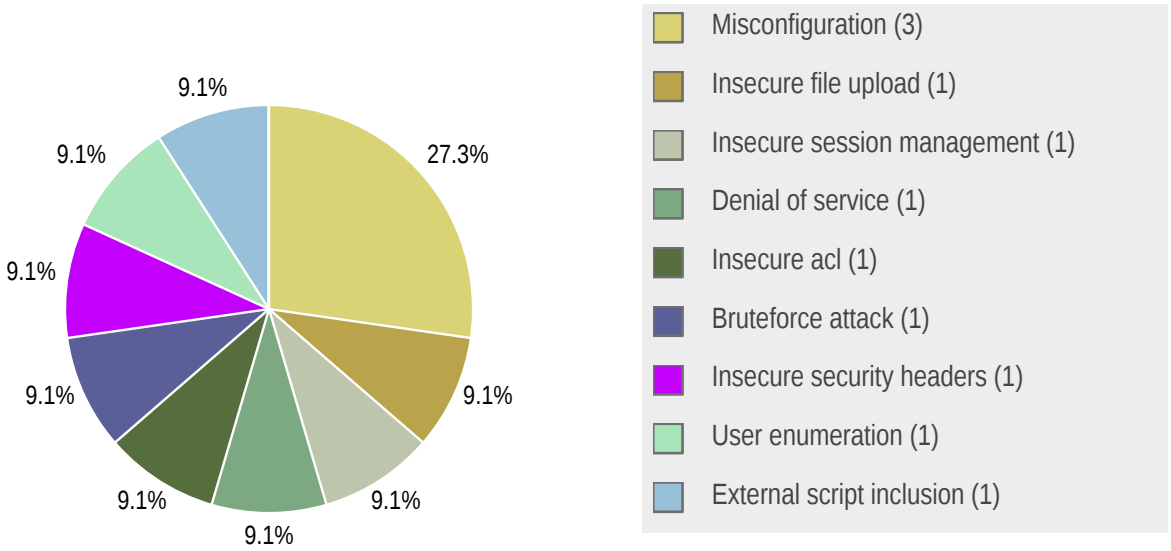
ID	Type	Description	Threat level
MAF-010	Insecure File Upload	Multiple security issues have been identified in the file upload functionality.	High
MAF-009	Insecure Session Management	Sessions are not properly managed, with no revocation on logout and no expiration.	Elevated
MAF-007	Denial of Service	The application allows contributors, editors and administrators to upload large compressed files.	Elevated
MAF-017	Insecure ACL	A viewer user has unauthorized access to view all conversion problems, including those of other users, and can initiate a conversion to 3MF format.	Moderate
MAF-008	Bruteforce Attack	The application lacks protection against password bruteforce attacks during login.	Moderate
MAF-005	Insecure Security Headers	Important security headers are missing or misconfigured.	Moderate
MAF-003	Misconfiguration	Docker container is running under the root user.	Moderate
MAF-002	User Enumeration	User enumeration is possible through the change password, reset password, and login forms.	Moderate

MAF-011	Misconfiguration	System paths in the library functionality allow administrators to configure root directories as the path.	Low
MAF-006	External Script Inclusion	The application includes the load of external scripts from a third-party.	Low
MAF-004	Misconfiguration	Several security misconfigurations were found in the Docker configuration.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
MAF-010	Insecure File Upload	• Validate and sanitize zip file paths during extraction to mitigate Zip Slip vulnerabilities. • To mitigate the race condition implement Locking Mechanisms, Atomic Operation and Generate a unique directory name for each operation. • Implement strict file size and type validation to prevent upload of malicious or oversized files.
MAF-009	Insecure Session Management	• Implement proper session management with token revocation on logout. • Set session expiration policies to automatically terminate sessions after a defined period of inactivity. • Use secure session storage mechanisms, such as secure cookies or server-side session storage. • Implement session monitoring and anomaly detection to identify and prevent suspicious session activities.
MAF-007	Denial of Service	• Implement file size and type checks both on the client-side and server-side to restrict the maximum allowed file size. This ensures that potentially harmful files are caught early in the upload process. • Limit the maximum decompressed size of uploaded files to prevent resource exhaustion. This can be achieved by setting a threshold for the decompressed data and aborting the process if this limit is exceeded. • Consider using a separate process or service to handle file decompression and validation. For instance, deploying this

		functionality within a dedicated Docker container can isolate potential threats and reduce the risk to the main application. • Implement rate limiting and monitoring mechanisms to detect and prevent abuse of the file upload functionality. By tracking the frequency and size of uploads, you can identify and mitigate suspicious activities promptly.
MAF-017	Insecure ACL	To mitigate this vulnerability, the following measures should be taken: • Implement access controls to restrict users with viewer access, from accessing problem issues of other users. • Limit the conversion capabilities of viewer users to prevent unauthorized conversions. • Implement logging and monitoring to detect and respond to potential security incidents.
MAF-008	Bruteforce Attack	• Implement account lockout mechanisms to temporarily lock accounts after a certain number of failed login attempts. • Implement rate limiting to restrict the number of login attempts from a single IP address within a given time frame. • Consider using CAPTCHAs or other challenge-response mechanisms to prevent automated attacks. This should also be added to other forms such as the password reset and registration forms to prevent abuse, such as flooding users' mailboxes and creating numerous accounts to exhaust system resources. • Enable multi-factor authentication (MFA) to add an extra layer of security to user accounts.
MAF-005	Insecure Security Headers	• Remove the obsolete X-XSS-Protection header and rely on modern browser built-in XSS protection such as CSP. • Enhance the CSP header to restrict content sources and prevent unauthorized access, e.g. adding `script-src`, `object-src` and `require-trusted-types-for`. • Implement the Strict-Transport-Security header to enforce secure connections. • Add the Permissions-Policy header to control which browser features can be used and by which origins.
MAF-003	Misconfiguration	• Configure Docker containers to run as a non-root user with least privileges.
MAF-002	User Enumeration	• Ensure uniform responses for both valid and invalid username/email entries to prevent information disclosure. • Implement rate limiting and CAPTCHA to prevent automated enumeration attempts. • Add extra time to these requests to mitigate time-based attacks.
MAF-011	Misconfiguration	• Implement a user-friendly file browser dialog or similar UI component that allows users to visually navigate and select the desired library folder. This provides a more intuitive and controlled way for users to specify the directory. • Configure an allowlist of directories that can be selected and displayed to the admin. This ensures that only pre-approved directories are available for selection. • Implement robust server-side path traversal checks to prevent users from crafting malicious paths that could bypass the intended directory restrictions.

		• Apply strict input validation and sanitization techniques to user-supplied paths to mitigate the risk of path injection attacks.
MAF-006	External Script Inclusion	• Ideally, host critical scripts locally to minimize the attack surface by maintaining control over the files within your own infrastructure. • Implement a Content Security Policy (CSP) to restrict the sources of external scripts. • Generate a unique nonce value for each request and include it in the `nonce` attribute of the script tags. This helps prevent unauthorized script execution by ensuring that only scripts with the correct nonce value are executed. By implementing these improvements and best practices, you can enhance the security of your application when including third-party scripts from Cloudflare or other sources.
MAF-004	Misconfiguration	Implement as many security recommendations as possible to reduce the attack vector. • Implement Docker multi-stage builds to minimize the attack surface. • Set security_opt: `no-new-privileges:true` to prevent privilege escalation. • Use a read-only filesystem (`read_only: true`) to protect against unauthorized modifications. • Drop unnecessary capabilities at container start using `cap_drop` and `cap_add` to follow the principle of least privilege. • Regularly review and update Docker configurations to align with security best practices.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2021) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Reconnaissance and Fingerprinting

We were able to gain information about the software and infrastructure through the following automated scans. Any relevant scan output will be referred to in the findings.

- Nmap – https://nmap.org
- Burp Suite Professional – https://portswigger.net/burp/pro
- Semgrep – https://semgrep.dev
- Trufflehog – https://github.com/trufflesecurity/trufflehog
- Testssl.sh – https://github.com/drwetter/testssl.sh

4 Findings

We have identified the following issues:

4.1 MAF-010 — Insecure File Upload

Vulnerability ID: MAF-010

Vulnerability type: Insecure File Upload

Threat level: High

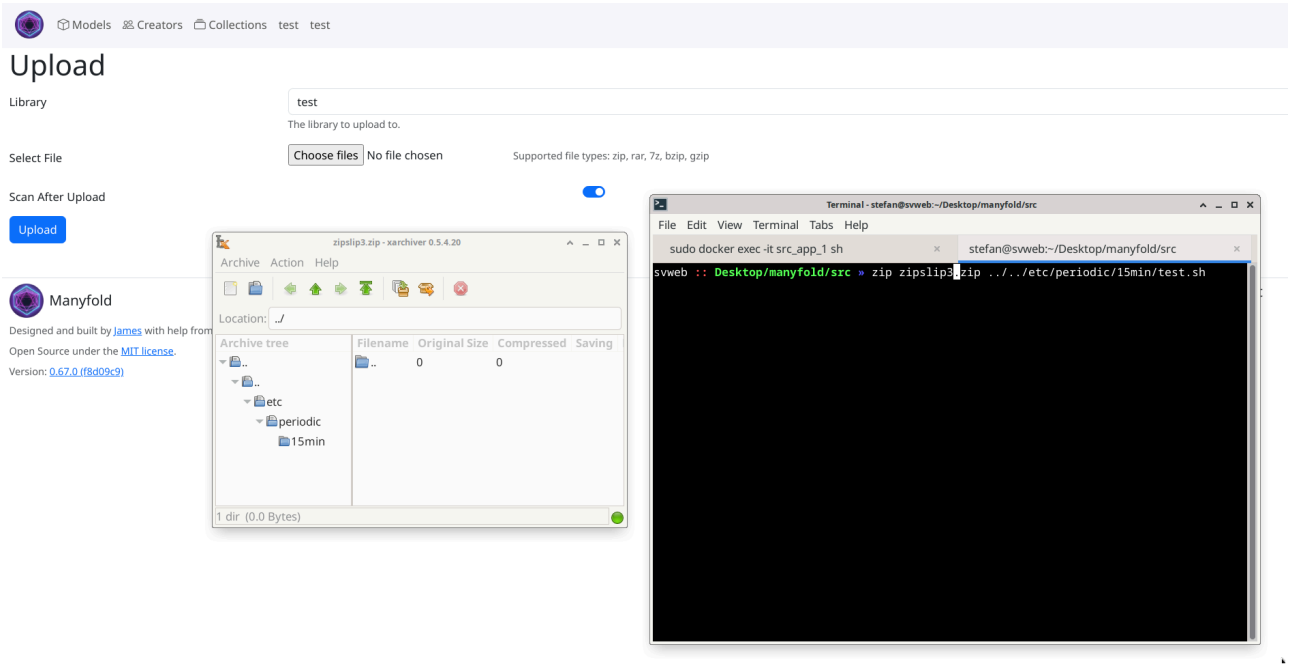
Description:

Multiple security issues have been identified in the file upload functionality.

Technical description:

Several vulnerabilities were found within the file upload functionality:

1. Zip Slip (unsafe zip extraction)



```
/usr/src/app # ls /etc/periodic/15min/
test.sh
```

Notice that the file `test.sh` is uploaded to `/etc/periodic/15min/test.sh` on the server. However, it did not execute because it lacked the necessary execute permissions.

Despite this, there are significant risks, as the ability to add or overwrite files in the filesystem remains. Running as root exacerbates these risks, allowing potential overwriting of critical files such as `authorized_keys`, `crontab`, and others. Consequently, any user with upload rights could exploit this vulnerability.

2. Race condition in directory existence checks

app > controllers > uploads_controller.rb

```
1   class UploadsController < ApplicationController
10    def create
16      end
17      redirect_to libraries_path, notice: t(".success")
18    end
19
20    private
21
22    def save_files(files, library_path)
23      files.select { |datafile| datafile != "" }.each { |datafile|
24        file_name_with_zip = datafile.original_filename
25        file_name = File.basename(file_name_with_zip, File.extname(file_name_with_zip))
26        dest_folder_name = library_path + file_name
27        if !Dir.exist?(dest_folder_name)
28          unzip(dest_folder_name, datafile)
29        end
30      }
31    end
```

The code checks if a directory does not exist (`if !Dir.exist?(dest_folder_name)`) and then proceeds to create it and unzip files into it. If two requests are processed concurrently for the same file name, both might pass the `Dir.exist` check before either has a chance to create the directory. This could lead to both trying to create the same directory and unzip files into it simultaneously, causing file write conflicts or data corruption.

3. Lack of file size/type validation

There is no comprehensive client-side and server-side validation of file size and type. Currently, the only client-side measure in place is specifying which file types are accepted, but this can be easily bypassed by users.

```
<div class="row mb-3 input-group">
  <%= form.label :files, t(".files.label"), class: "col-sm-2 col-form-label" %>
  <div class='col-sm-10 ps-0'>
    <%= form.file_field :files, {
          multiple: true,
          accept: "application/zip,application/x-rar-compressed,application/x-7z-compressed,
          application/x-bzip2,application/x-bzip,application/gzip"
        } %>
    <span class="form-text"><%= t ".files.help", types: safe_join(%w[zip rar 7z bzip gzip],
    ", ") %></span>
  </div>
</div>
```

Impact:

These vulnerabilities can lead to code execution, overwriting critical files, and denial of service.

Recommendation:

- Validate and sanitize zip file paths during extraction to mitigate Zip Slip vulnerabilities.
- To mitigate the race condition implement Locking Mechanisms, Atomic Operation and Generate a unique directory name for each operation.
- Implement strict file size and type validation to prevent upload of malicious or oversized files.

4.2 MAF-009 — Insecure Session Management

Vulnerability ID: MAF-009

Vulnerability type: Insecure Session Management

Threat level: Elevated

Description:

Sessions are not properly managed, with no revocation on logout and no expiration.

Technical description:

User sessions are not revoked after logout, and there is no session expiration policy in place. This can lead to unauthorized access if session tokens are reused or intercepted.

Impact:

Sessions can be hijacked or reused, leading to unauthorized access and data breaches.

Recommendation:

- Implement proper session management with token revocation on logout.
- Set session expiration policies to automatically terminate sessions after a defined period of inactivity.
- Use secure session storage mechanisms, such as secure cookies or server-side session storage.
- Implement session monitoring and anomaly detection to identify and prevent suspicious session activities.

4.3 MAF-007 — Lack of Zip Decompression Bomb Protection

Vulnerability ID: MAF-007

Vulnerability type: Denial of Service

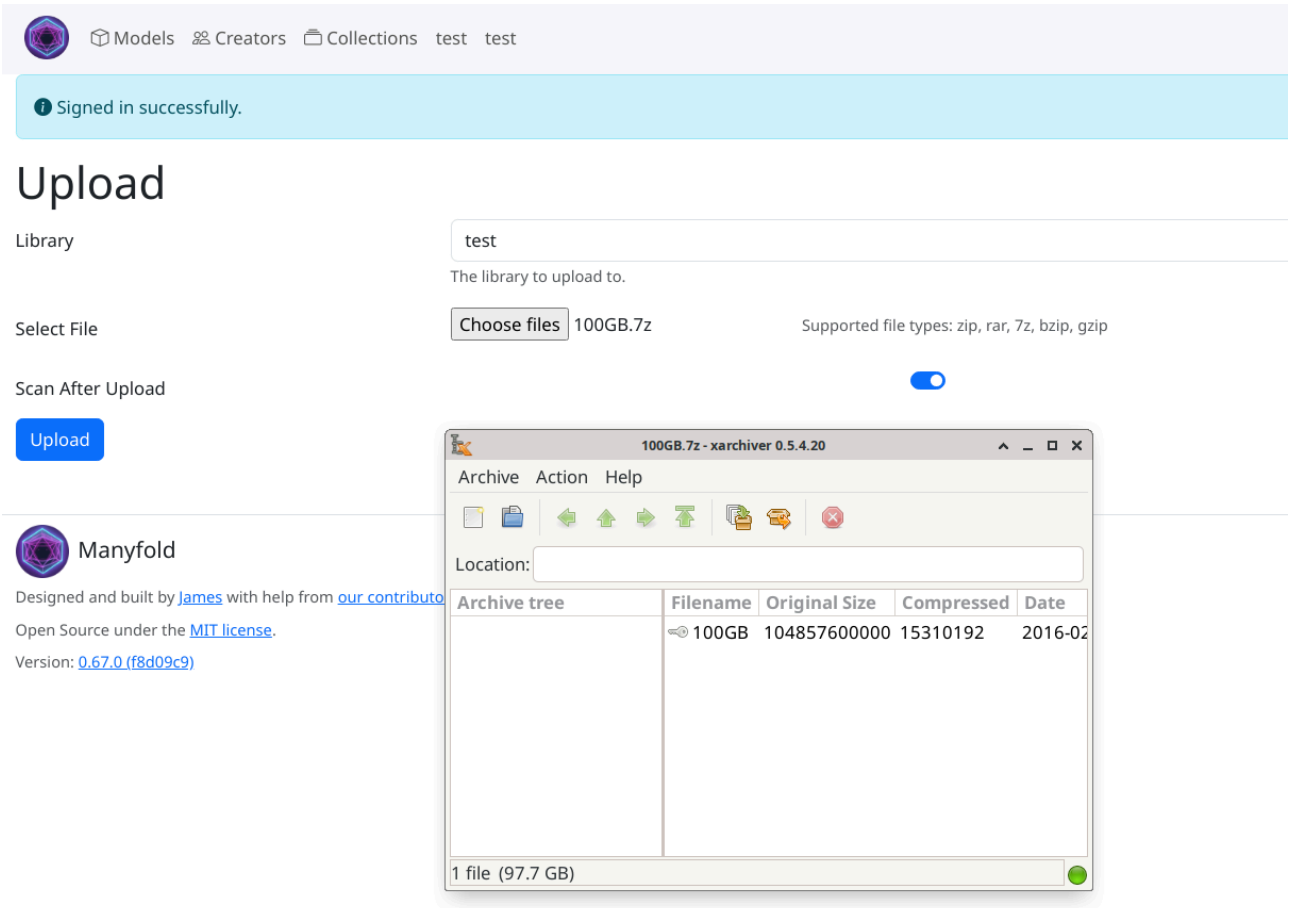
Threat level: Elevated

Description:

The application allows contributors, editors and administrators to upload large compressed files.

Technical description:

The application allows users to upload large compressed files, such as a 15MB 7z file that decompresses into a much larger 100GB file.




```
/tmp # ls -ll 100GB
total 0
-rw-r--r--    1 root     root     104857600000 May 30 23:58 100GB
/tmp #
```

Currently, there is no built-in server-side functionality to prevent this. This vulnerability can lead to a Zip Bomb or Decompression Bomb attack, where a maliciously crafted zip file exhausts system resources like memory and disk space during decompression, potentially causing a denial of service.

It is important to note that, by default, newly registered users do not have file upload permissions when user registration is enabled. Only users with elevated roles, such as editors, contributors, and administrators, are permitted to upload files.

Impact:

Large file uploads can exhaust server resources, leading to potential denial of service.

Recommendation:

- Implement file size and type checks both on the client-side and server-side to restrict the maximum allowed file size. This ensures that potentially harmful files are caught early in the upload process.
- Limit the maximum decompressed size of uploaded files to prevent resource exhaustion. This can be achieved by setting a threshold for the decompressed data and aborting the process if this limit is exceeded.
- Consider using a separate process or service to handle file decompression and validation. For instance, deploying this functionality within a dedicated Docker container can isolate potential threats and reduce the risk to the main application.
- Implement rate limiting and monitoring mechanisms to detect and prevent abuse of the file upload functionality. By tracking the frequency and size of uploads, you can identify and mitigate suspicious activities promptly.

4.4 MAF-017 — Viewers have unrestricted access to conversion problems and 3MF conversion

Vulnerability ID: MAF-017

Vulnerability type: Insecure ACL

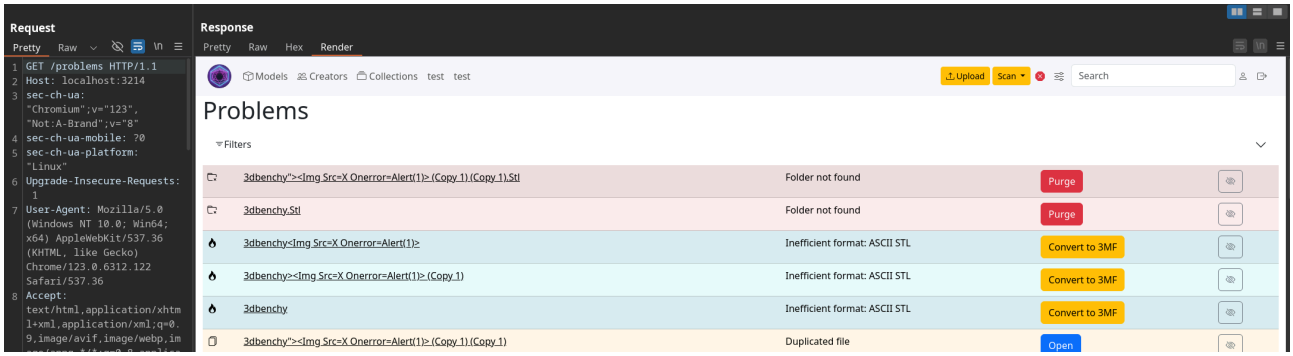
Threat level: Moderate

Description:

A viewer user has unauthorized access to view all conversion problems, including those of other users, and can initiate a conversion to 3MF format.

Technical description:

During the penetration test, it was observed that a user with viewer role, can access and view all conversion problems, including those belonging to other users. This is a security concern as it allows unauthorized access to potential sensitive information.



Furthermore, the viewer user can also initiate a conversion to 3MF format, which could lead to unintended changes to the original data.

Impact:

An attacker with viewer user privileges can exploit this vulnerability to access potential sensitive information and potentially disrupt the conversion process. This could lead to data breaches and unauthorized changes to data.

Recommendation:

To mitigate this vulnerability, the following measures should be taken:

- Implement access controls to restrict users with viewer access, from accessing problem issues of other users.
- Limit the conversion capabilities of viewer users to prevent unauthorized conversions.
- Implement logging and monitoring to detect and respond to potential security incidents.

4.5 MAF-008 — Lack of Password Bruteforce Protection

Vulnerability ID: MAF-008

Vulnerability type: Bruteforce Attack

Threat level: Moderate

Description:

The application lacks protection against password bruteforce attacks during login.

Technical description:

The login functionality does not have measures to prevent password brute-force attacks. This leaves the application vulnerable to automated attacks by trying multiple password combinations to gain unauthorized access.

Impact:

Attackers can attempt numerous password combinations, increasing the risk of account compromise.

Recommendation:

- Implement account lockout mechanisms to temporarily lock accounts after a certain number of failed login attempts.
- Implement rate limiting to restrict the number of login attempts from a single IP address within a given time frame.
- Consider using CAPTCHAs or other challenge-response mechanisms to prevent automated attacks. This should also be added to other forms such as the password reset and registration forms to prevent abuse, such as flooding users' mailboxes and creating numerous accounts to exhaust system resources.
- Enable multi-factor authentication (MFA) to add an extra layer of security to user accounts.

4.6 MAF-005 — Missing or Insecure Security Headers

Vulnerability ID: MAF-005

Vulnerability type: Insecure Security Headers

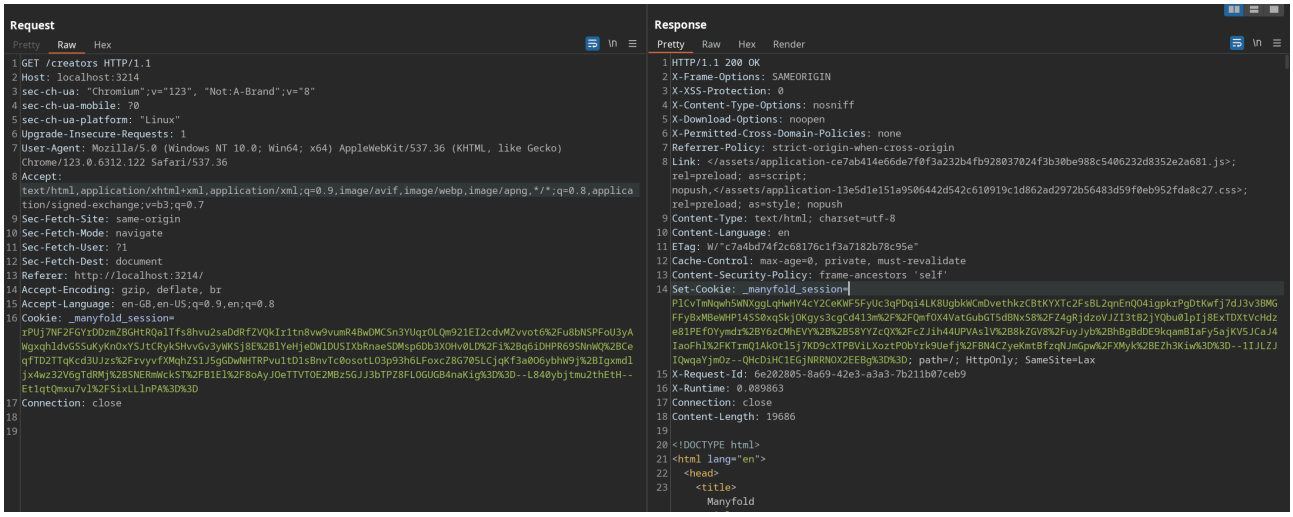
Threat level: Moderate

Description:

Important security headers are missing or misconfigured.

Technical description:

The application lacks essential security headers or has these misconfigured:



- X-XSS-Protection is set to 0
- The CSP header is minimal (frame-ancestors 'self')
- Lack of Strict-Transport-Security header
- Missing Permissions-Policy

Impact:

Missing or misconfigured security headers might expose the application to XSS attacks, clickjacking, and other security risks.

Recommendation:

- Remove the obsolete X-XSS-Protection header and rely on modern browser built-in XSS protection such as CSP.
- Enhance the CSP header to restrict content sources and prevent unauthorized access, e.g. adding `script-src`, `object-src` and `require-trusted-types-for`.
- Implement the Strict-Transport-Security header to enforce secure connections.
- Add the Permissions-Policy header to control which browser features can be used and by which origins.

4.7 MAF-003 — Docker Container Running as Root

Vulnerability ID: MAF-003

Vulnerability type: Misconfiguration

Threat level: Moderate

Description:

Docker container is running under the root user.

Technical description:

The Docker container is configured to run as the root user. This can lead to security issues, as files and folders created within the container have root ownership, increasing the risk of privilege escalation attacks. This is a known issue and the developer has plans to change this.

Impact:

Running containers as root increases the attack surface and potential for privilege escalation within the container.

Recommendation:

- Configure Docker containers to run as a non-root user with least privileges.

4.8 MAF-002 — User Enumeration in Password and Login Forms

Vulnerability ID: MAF-002

Vulnerability type: User Enumeration

Threat level: Moderate

Description:

User enumeration is possible through the change password, reset password, and login forms.

Technical description:

The HTTP response of these requests reveals whether an email is taken or a username already exists. This information disclosure allows attackers to enumerate valid usernames and emails.

Impact:

Attackers can enumerate valid usernames and email addresses, facilitating targeted attacks such as phishing or brute force.

Recommendation:

- Ensure uniform responses for both valid and invalid username/email entries to prevent information disclosure.
- Implement rate limiting and CAPTCHA to prevent automated enumeration attempts.
- Add extra time to these requests to mitigate time-based attacks.

4.9 MAF-011 — Library Path Misconfiguration

Vulnerability ID: MAF-011

Vulnerability type: Misconfiguration

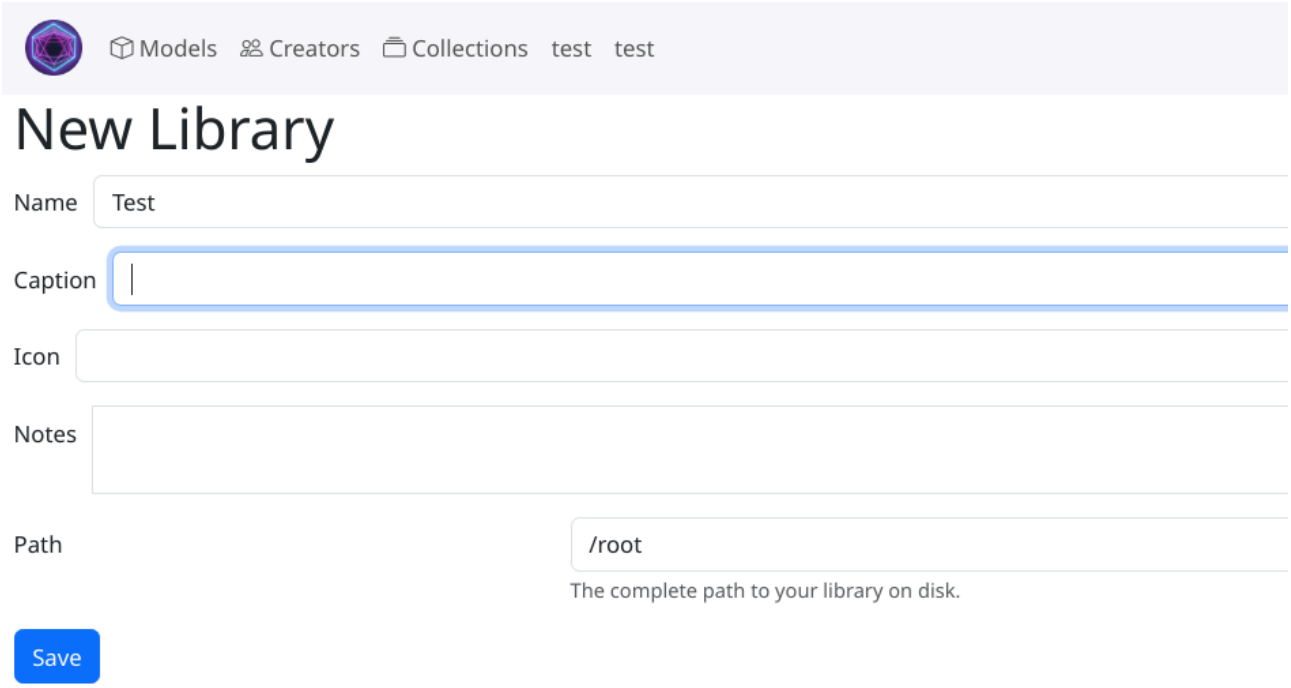
Threat level: Low

Description:

System paths in the library functionality allow administrators to configure root directories as the path.

Technical description:

The library system functionality allow administrators to set paths to root directories:



Paths should be restricted to only allow specific directories to minimize the attack vector.

Impact:

Allowing access to root directories significantly increases the risk of unauthorized access, data breaches, and potential system compromises. Attackers could exploit this misconfiguration to gain elevated privileges and access sensitive files or directories.

Recommendation:

- Implement a user-friendly file browser dialog or similar UI component that allows users to visually navigate and select the desired library folder. This provides a more intuitive and controlled way for users to specify the directory.
- Configure an allowlist of directories that can be selected and displayed to the admin. This ensures that only pre-approved directories are available for selection.
- Implement robust server-side path traversal checks to prevent users from crafting malicious paths that could bypass the intended directory restrictions.
- Apply strict input validation and sanitization techniques to user-supplied paths to mitigate the risk of path injection attacks.

4.10 MAF-006 — External Script Inclusion

Vulnerability ID: MAF-006

Vulnerability type: External Script Inclusion

Threat level: Low

Description:

The application includes the load of external scripts from a third-party.

Technical description:

Including third-party scripts, such as those from Cloudflare, can pose security risks if the scripts are compromised.

It is essential to ensure the integrity of these scripts and monitor for any changes.

Example of the included JavaScript in the code:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js"
 integrity="sha512-894YE6QWD5I59HgZOGReFYm4dnWc1Qt5NtvYSaNcOP+u1T9qYdvdihz0PPSiiqn/
+/3e7Jo4EaG7TubfWGUrMQ==" crossorigin="anonymous" referrerpolicy="no-referrer" nonce=""></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/selectize.js/0.13.6/
js/standalone/selectize.min.js" integrity="sha512-pgmLgtHvorzxpKra2mmibwH/
RDAVMlOuqU98ZjnyZrOZxgAR8hwL8A02hQFWEK25V40/9yPYb/Zc+kyWMplgaA==" crossorigin="anonymous"
 referrerpolicy="no-referrer" nonce=""></script>
```

Note that the integrity attribute and anonymous crossorigin have already been configured in the original code snippet, which helps protect against malicious attacks by verifying the integrity of the downloaded scripts. However, further improvements can be made by configuring a more comprehensive Content-Security-Policy (CSP) and generating a unique nonce value for each request to include in the script tags. These enhancements will provide additional layers of security for your web application.

Impact:

External scripts can be a vector for introducing malicious code if compromised, potentially leading to site-wide security breaches.

Recommendation:

- Ideally, host critical scripts locally to minimize the attack surface by maintaining control over the files within your own infrastructure.
- Implement a Content Security Policy (CSP) to restrict the sources of external scripts.

- Generate a unique nonce value for each request and include it in the `nonce` attribute of the script tags. This helps prevent unauthorized script execution by ensuring that only scripts with the correct nonce value are executed.

By implementing these improvements and best practices, you can enhance the security of your application when including third-party scripts from Cloudflare or other sources.

4.11 MAF-004 — Insecure Docker Configuration

Vulnerability ID: MAF-004

Vulnerability type: Misconfiguration

Threat level: Low

Description:

Several security misconfigurations were found in the Docker configuration.

Technical description:

The Docker configuration has room for improvement.

Docker-Compose:

```
version: "3"

services:
  app:
    image: ghcr.io/manyfold3d/manyfold:latest
    ports:
      - 3214:3214
    volumes:
      - /path/to/your/libraries:/libraries
    environment:
      SECRET_KEY_BASE: a_nice_long_random_string
      REDIS_URL: redis://redis:6379/1
      DATABASE_URL: postgresql://manyfold:password@db/manyfold?pool=5
#               or
#      DATABASE_HOST: db
#      DATABASE_USER: manyfold
#      DATABASE_PASSWORD: password
#      DATABASE_NAME: manyfold

      # For details of other optional environment variables, including features such
      # as multiuser mode, visit https://manyfold.app/sysadmin/configuration.html
    depends_on:
      - db
      - redis
    networks:
      - manyfold
    links:
      - db
      - redis

  db:
    image: postgres:15
    volumes:
      - db_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: manyfold
      POSTGRES_PASSWORD: password
    restart: on-failure
    networks:
      - manyfold

  redis:
    image: redis:7
    restart: on-failure
    networks:
      - manyfold

volumes:
  db_data:


networks:
  manyfold:
```

Dockerfile:

```
FROM ruby:3.3.0-alpine3.18 AS build

RUN apk add --no-cache tzdata alpine-sdk postgresql-dev nodejs yarn xz libarchive mesa-gl glfw
RUN gem install foreman

ARG APP_VERSION
ARG GIT_SHA

ENV PORT=3214
ENV RACK_ENV=production
ENV RAILS_ENV=production
ENV NODE_ENV=production
ENV RAILS_SERVE_STATIC_FILES=true
ENV APP_VERSION=${APP_VERSION}
ENV GIT_SHA=${GIT_SHA}

WORKDIR /usr/src/app

COPY package.json .
COPY yarn.lock .
RUN yarn config set network-timeout 600000 -g
RUN yarn install

RUN gem install bundler -v 2.4.13
RUN bundle config set --local deployment 'true'
RUN bundle config set --local without 'development test'
COPY .ruby-version .
COPY Gemfile* ./
RUN bundle install

COPY . .
RUN \
  DATABASE_URL="nulldb://user:pass@localhost/db" \
  SECRET_KEY_BASE="placeholder" \
  bundle exec rake assets:precompile

EXPOSE 3214
ENTRYPOINT ["bin/docker-entrypoint.sh"]
CMD ["foreman", "start"]
```

The following issues were found:

- No utilization of multi-stage builds.
- Does not prevent containers from escalating privileges.
- Does not use a read-only filesystem.
- Does not drop unnecessary capabilities at container start.

Impact:

These misconfigurations increase the risk of privilege escalation, persistent vulnerabilities, and potential security breaches.

Recommendation:

Implement as many security recommendations as possible to reduce the attack vector.

- Implement Docker multi-stage builds to minimize the attack surface.
- Set security_opt: `no-new-privileges:true` to prevent privilege escalation.
- Use a read-only filesystem (`read_only: true`) to protect against unauthorized modifications.
- Drop unnecessary capabilities at container start using `cap_drop` and `cap_add` to follow the principle of least privilege.
- Regularly review and update Docker configurations to align with security best practices.

5 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends, or represents issues that are not directly security-related.

5.1 NF-015 — Absence of SQL and Command Injection Vulnerabilities

The security assessment did not uncover any SQL injection or command injection vulnerabilities within the application. This suggests that the development team has implemented proper input validation and sanitization techniques to prevent malicious actors from manipulating database queries or executing unauthorized commands.

5.2 NF-014 — No Cross-Site Scripting Vulnerabilities Identified

During the security assessment, no instances of cross-site scripting (XSS) or DOM-based XSS were discovered. However, as an additional precautionary measure, it is recommended to review and enhance the configuration of security headers. Properly configured security headers serve as an extra layer of defense against potential XSS attacks, further strengthening the application's security posture.

5.3 NF-013 — Up-to-Date Package Management

An assessment of the packages utilized in the project demonstrated that all dependencies were up-to-date. Maintaining updated packages is crucial for ensuring the security of the application, as older versions may contain known vulnerabilities. By keeping the packages current, the development team has effectively reduced the risk of potential security breaches introduced by outdated dependencies.

5.4 NF-012 — No Secrets Exposed in GitHub Repository

A review of the GitHub repository, including code commits and issues, revealed no instances of exposed secrets. This indicates that the development team has been diligent in following best practices for protecting sensitive information, such as credentials or confidential data, from being inadvertently leaked through the repository.

6 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

7 Conclusion

We discovered 1 High, 2 Elevated, 5 Moderate and 3 Low-severity issues during this penetration test.

The security assessment has highlighted several critical areas that require immediate attention to mitigate risks and enhance the overall security posture of the application. High-severity issues, such as vulnerabilities in the file upload functionality (Zip Slip and lack of file size/type validation) and insecure session management, pose significant risks of unauthorized access and system compromise. Immediate remediation of these vulnerabilities is essential to prevent potential code execution, critical file overwriting, and session hijacking.

Elevated severity issues, including the lack of protection against zip decompression bombs and missing or insecure security headers, also demand prompt action. These vulnerabilities could lead to denial of service attacks, XSS attacks, and clickjacking. Implementing protection mechanisms and properly configuring security headers will mitigate these risks. Additionally, introducing password brute force protection during login is crucial to safeguard against account compromise.

Moderate and low-severity issues, while individually less critical, collectively weaken the security framework. Addressing user enumeration vulnerabilities, securing external script inclusion, and rectifying various misconfigurations such as insecure Docker configurations, Docker containers running as root, and library path misconfiguration are necessary steps.

On a positive note, the application demonstrates several strengths in its security measures. Notably, no Cross-Site Scripting (XSS) vulnerabilities or privilege escalation issues were identified, and there was an absence of SQL and command injection vulnerabilities. Additionally, the application benefits from up-to-date package management and no secrets were exposed in the GitHub repository. These practices significantly contribute to the security of the application. Maintaining these will continue to enhance the application's security posture.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Stefan Vink	Stefan is an IT professional with a passion for IT security and automation. With 20 years hands-on experience in a diverse range of IT roles such as automation / scripting / monitoring / web development / system and network management in Windows and Linux environments. He has worked for organisations such as the Central Bank of the Netherlands (DNB), is MCITP, CCNA, LPIC, OSCP certified, and has passed the CISSP exam. He loves to travel, hike, play tennis & chess, automation, and lives with his wife and kids in Melbourne, Australia.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.