This file contains a summary of most tampering possibilities and how they are (or are not) countered in my application

Threats

1. Tampering inside the uploaded model.keras or model.h5 file

Threat A malicious actor could embed a `Lambda` layer inside a TensorFlow/Keras model file (`.keras` or `.h5`). The `Lambda` layer allows arbitrary Python code to be serialized and executed when the model is loaded. This could lead to remote code execution (RCE), enabling attackers to execute harmful commands on the system, such as creating reverse shells or deleting files.

In the context of this platform, such an attack could be used to tamper with or disable the measurement tools responsible for generating a trustworthy AIBoM. By compromising these tools, an attacker could falsify the recorded artifacts, manipulate the training process metadata, or bypass integrity checks, undermining the trustworthiness of the entire system.

Countermeasure in the application

- **Safe Mode Loading:** The application uses TensorFlow's `safe_mode=True` option when loading models. According to the TensorFlow documentation, this option ensures that unsafe Lambda deserialization is disallowed, preventing arbitrary code execution during model loading. This argument is only applicable to the Keras v3 model format, so only these types of model files are allowed. The application enforces this setting by default:

  ```
  model = tf.keras.models.load_model(model_path, safe_mode=True)
  ```

- **Restricted Model Loading Environment:** Models are loaded in restricted Docker containers configured with only the necessary permissions. This minimizes the impact of potential malicious code by isolating the model loading process. Additionally, this setup can be extended with the use of sandboxes, virtual machines, or lightweight virtualization technologies such as AWS Firecracker to further enhance security and isolation.

- **Model Integrity Verification:** When the AIBOM is generated, all materials, including the model, are hashed and included in the BOM. This ensures that any injected malicious code in the model file is also hashed and recorded.

Remaining risks

- If a malicious actor finds a vulnerability in TensorFlow's `safe_mode` implementation, they could potentially bypass the safeguard.
- The system cannot detect malicious intent embedded in the model's architecture or weights (e.g., backdoors or adversarial triggers). External auditing of the model is required.

2. Tampering with the dataset (.zip or .csv) or definition file

Threat

A malicious actor could tamper with the dataset files (e.g., `.zip` or `.csv`) or the dataset definition file. This could involve injecting malicious data, altering the dataset structure, or modifying the dataset definition to cause unexpected behavior during training. Such tampering could lead to corrupted models, biased results, or even system crashes.

Countermeasure in the application

- **Dataset Validation:** The application validates the structure and content of the dataset files before use. For example:
  - `.zip` files are carefully validated and extracted securely to ensure they contain the expected files and structure. This includes checks for malicious content such as **ZIP bombs** to prevent resource exhaustion attacks.
  - `.csv` files are checked for consistency with the dataset definition (e.g., column names, data types).
- **Dataset Definition Validation:** The dataset definition file is parsed and validated to ensure it adheres to the expected schema. Any inconsistencies or unexpected modifications are flagged, and the process is halted.
- **Controlled Environment:** Dataset processing is performed in an isolated Docker container, ensuring that any potential malicious data cannot affect the broader system. This isolation limits the scope of any potential attack and protects the integrity of the platform.

Remaining risks

- If the input files themselves are malicious but valid (e.g., a biased dataset), your system cannot detect this. This requires external auditing or dataset validation mechanisms.

---

3. Tampering with the Training Process

A malicious actor could interfere with the training process by modifying training parameters, injecting malicious code into the training logic, or altering the environment in which the training occurs. This could lead to corrupted models, biased results, or compromised outputs.

Countermeasure in the application

- **Container Isolation:** Training jobs are executed in isolated Docker containers, preventing developers from directly accessing the system or modifying the training process.
- **In-Toto Verification:** The run_training step is attested using in-toto, and the `.link` file includes hashes of all input and output artifacts. Users can verify the `.link` file against the signed layout to ensure the training process followed the expected steps.
- **CycloneDX BOM:** The CycloneDX BOM includes metadata about the training environment (e.g., Python version, TensorFlow version), ensuring the environment is consistent and trusted.

Remaining risks

- If the developer gains access to the worker container (e.g., through a vulnerability in the container runtime), they could tamper with the training process. This risk is partially mitigated by:
    - **Read-Only Containers with Temporary Storage:** The `docker-compose.yml` file specifies `read_only: true` for workers to limit write access within the container. Additionally, a `tmpfs` mount is used to allow temporary file storage in memory, ensuring that necessary temporary files can still be created without compromising the container's read-only nature. For more details, refer to the Docker Compose documentation on services and the Docker documentation on tmpfs storage.
    - **Capability Restrictions:** Dropping all capabilities (`cap_drop: ALL`) and only allowing `NET_BIND_SERVICE` reduces the attack surface. See the Docker documentation on capabilities for more information.
    - **Non-Root User:** Worker containers are configured to run as a non-root user by specifying the `user` directive in the `docker-compose.yml` file. This minimizes the impact of potential exploits by restricting access to privileged operations. For more details, refer to the Docker documentation on user namespaces.

---

4. Tampering with the Generated Artifacts

Threat

A malicious actor could tamper with the generated artifacts (e.g., trained models, logs, metrics, AIBoM files) after they are created but before they are shared with users. This could lead to compromised models, falsified metrics, or altered AIBoMs.

Countermeasure in the application

- **Hash-Based Integrity Verification:** All generated artifacts are hashed, and these hashes are included in the AIBoM and in-toto .link files. Users can verify the integrity of the artifacts by comparing their hashes with the recorded values.
- **Secure Storage:** Artifacts are stored in a secure MinIO bucket with restricted access to prevent unauthorized modifications.
- **Cryptographic Signing:** The AIBoM and in-toto .link files are cryptographically signed to ensure their authenticity and integrity.

Remaining risks

- If the storage system (e.g., MinIO) is compromised, attackers could tamper with the artifacts despite secure storage. This is partially mitigated by:
    - The API provides an endpoint to check the hashes of the MinIO artifacts with the recorded hashes. If they fail that means someone tampered in the MinIO storage. However they can not update the link file to match their tampered artifacts because they do not have access to the platform private key to correctly resign the link file.
- The system does not currently support real-time monitoring of artifact access or modifications.

---

5. Tampering with the AIBoM

Threat

A malicious actor could tamper with the AIBoM to falsify information about the training process, inputs, or outputs. This could mislead users into trusting a compromised model.

Countermeasure in the application

- **Cryptographic Signing:** The AIBoM is signed using the system's private key. Users can verify the signature using the public key to ensure the AIBoM has not been tampered with.
- **Hash Verification:** The AIBoM includes hashes of all input and output artifacts, allowing users to verify their integrity.

Remaining risks

- If the private key used for signing the AIBoM is compromised, attackers could generate falsified AIBoMs that appear legitimate.
- The system does not currently enforce key rotation or hardware-based key storage, which could reduce the risk of key compromise.

---

6. Unauthorized Access to Artifacts or System Components

Threat

A malicious actor could gain unauthorized access to artifacts (e.g., datasets, models, logs) or system components (e.g., API, workers, storage). This could lead to data breaches, tampering, or system compromise.

Countermeasure in the application

- **Authentication and Authorization:** The system uses Azure OAuth for secure API calls, ensuring that only authenticated and authorized users can access the system.
- **Access Control:** MinIO buckets and other system components can be configured with strict access controls when deployed to prevent unauthorized access.

Remaining risks

- If an attacker exploits a zero-day vulnerability in the authentication or authorization mechanisms, they could gain unauthorized access.
- The system does not yet have alerting for suspicious activity implemented.

---

7. Bypassing AIBOM generation

Threat

A malicious actor could attempt to bypass the AIBoM generation process to avoid recording the training process or its artifacts. This could lead to untraceable or unverified models.

Countermeasure in the application

- **Enforced AIBoM Generation:** The system automatically generates an AIBoM for every training job. Users cannot bypass this step.
- **Controlled API:** All interactions with the system are performed through a secure API, ensuring that users cannot interfere with the AIBoM generation process.

Remaining risks

- If an attacker exploits a vulnerability in the API or worker logic, they could potentially bypass the AIBoM generation process.

---

8. Exploiting Vulnerabilities in the Platform

Threat

A malicious actor could exploit vulnerabilities in the platform (e.g., API, worker containers, storage) to gain unauthorized access, execute malicious code, or compromise the system.

Countermeasure in the application

- **Regular Vulnerability Scanning:** The system scans container images and dependencies for vulnerabilities using tools like Trivy.
- **Read-Only Containers:** Worker containers are configured as read-only to limit the impact of potential exploits.
- **Capability Restrictions:** Containers drop all capabilities (cap_drop: ALL) and only allow minimal capabilities (e.g., NET_BIND_SERVICE) to reduce the attack surface.

Remaining risks

- New vulnerabilities in dependencies or container images could be exploited before they are patched.

---

9. Providing a Fake .link and/or AIBOM file

Threat

A malicious actor could provide a fake .link file or AIBoM to mislead users into trusting a compromised model or falsified training process.

Countermeasure in the application

- **Cryptographic Verification:** Users can verify the authenticity of .link files and AIBoMs using the system's public key. Any tampering or forgery will result in a failed verification.
- **Hash Verification:** The .link file and AIBoM include hashes of all input and output artifacts, ensuring that users can verify their integrity.

Remaining risks

- If the public key used for verification is replaced or tampered with, users could be misled into trusting fake files.
- The system does not currently provide a mechanism for users to verify the authenticity of the public key itself (e.g., through a trusted certificate authority).

Future work

Recommendations for further improving trustability:

- **Use Hardware Security Modules (HSMs):** Store private keys in HSMs to enhance security and reduce the risk of key compromise.
- **Key Rotation and Management:** Implement regular key rotation policies and integrate with secure key management systems to minimize the impact of key compromise.
- **Code Integrity Verification:** Include hashes of the worker container code, API code, and other critical components in the AIBoM to detect unauthorized modifications.
- **Expand the Scanner Container:** Extend the scanner container's functionality to scan not only worker containers but also the API, storage systems, and other platform components for vulnerabilities.
- **Real-Time Monitoring and Alerting:** Implement real-time monitoring and alerting for suspicious activity, such as unauthorized access to containers, storage systems, or API endpoints.
- **Public Key Authenticity Verification:** Provide a mechanism for users to verify the authenticity of the public key (e.g., through a trusted certificate authority or blockchain-based verification).
- **Runtime Monitoring of Containers:** Introduce runtime monitoring for containers to detect and respond to active exploitation attempts in real time.
- **Enhanced Dataset Validation:** Develop advanced mechanisms for detecting malicious but valid datasets (e.g., biased or poisoned data) using external auditing or AI-based validation tools.
- **User Education and Documentation:** Provide detailed documentation and training for users on how to verify AIBoMs, `.link` files, and artifacts to ensure proper usage of the system's trustability features.