Project Phase 2 – Security Analysis

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do. Accountable together – We are Purdue.

(On group submissions, have each team member type their name).

Type or sign your names: __Anjali Vanamala, Shaantanu Sriram, Andrew Diab, Pryce Tharpe

Write today's date: ___12/14/2025____

Assignment

This document provides a template for your system's security analysis. You are welcome to add content beyond what is listed; the template is designed to help ensure you cover all crucial components.

The terms used in this section are defined here.

Security requirements.

Here, list the *security* requirements of your system, aligned with the six security properties defined by the STRIDE article. These requirements may be unique to your system, depending on which features you implemented. It is possible that you will not have a requirement associated with every security property.

Confidentiality

- [All systems]: Observers along the network cannot directly observe client-server interactions.
- API keys are not exposed at any point.
- Logs and performance metrics shouldn't expose other data

Integrity

- Contents of models shouldn't be altered during ingestion or retrieval
- Outside dashboard actions, like performance measurement, should not change the data in any way.

Availability

- System should be able to handle the performance measurement test: 100 simultaneous downloads and 500 models in registry
- System health dashboard should serve live information

Authentication

- There must exist a default user

Authorization

- Only allowed operations should be exposed at endpoints.

Nonrepudiation

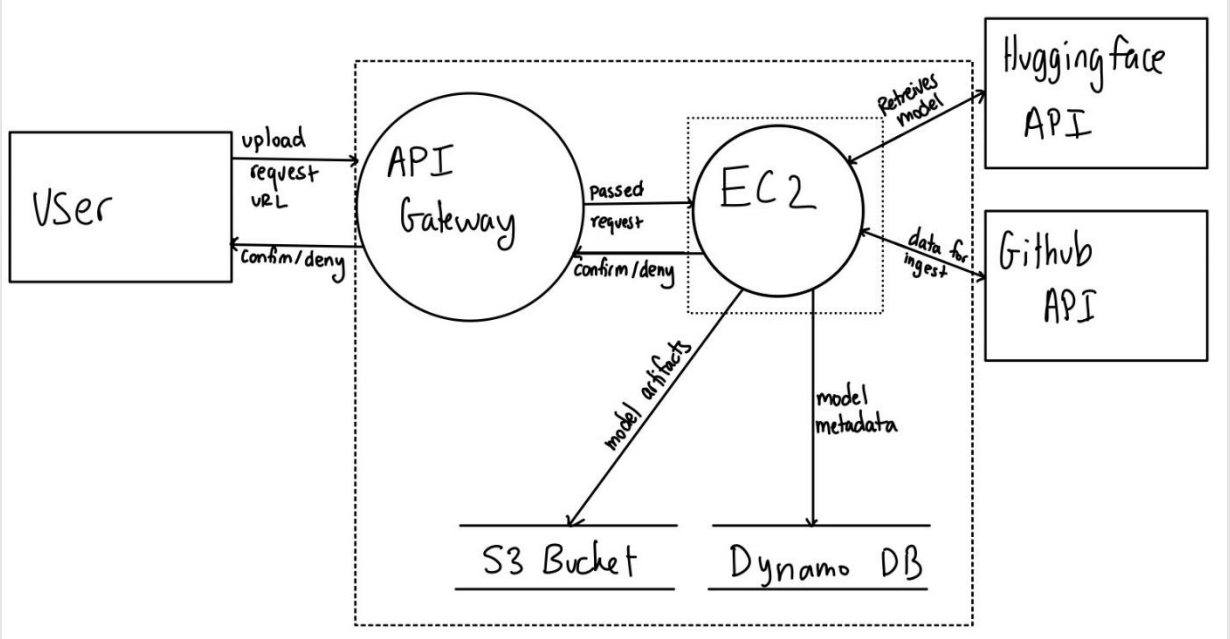
- Logging should be thorough
- Performance tests should be logged

System model via DFDs

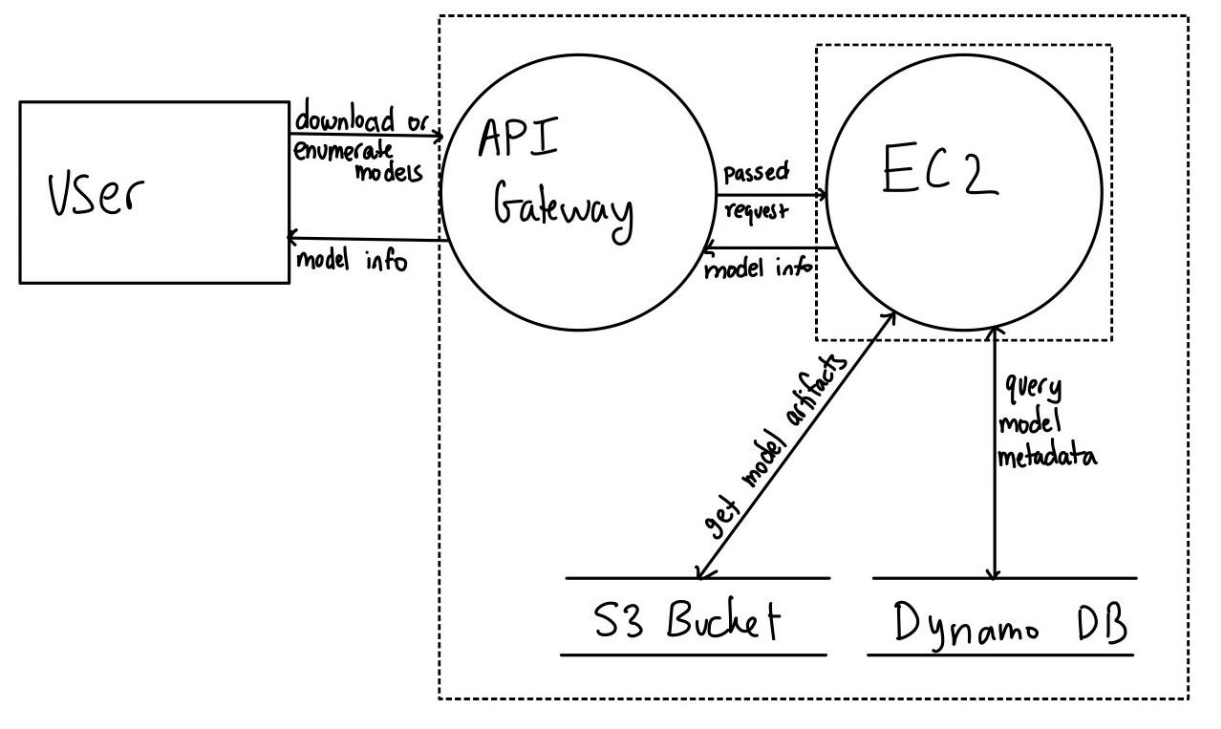
Present one or more data-flow diagrams of your deployed system. You can use some drawing tools such as https://app.diagrams.net/. A whiteboard picture is also fine, but use the correct symbols please – process, data sink, etc.

- You may provide multiple DFDs to capture different aspects of the system (e.g. one per feature or feature group).
- You may wish to indicate multiple trust boundaries, e.g. for different classes of users or for external components.
- Each diagram should indicate **at least** the following entities: data flow; data store; process; trust boundary. You may include interactors and multi-process if needed.

DFD for model creation
Larger trust boundary -> external users/services
Inner trust boundary-> our deployed code verses other AWS services



DFD for download model or enumerate models



Threats

For each trust boundary indicated in the DFDs, describe the nature of the untrusted party involved (examples: "outsider threat [e.g. external hacker]" or "insider threat [e.g. ACME employee with valid credentials]" or "infrastructure provider threat [AWS]"), why the location of the trust boundary is appropriate, and a potential attack that could occur were this trust boundary not secured.

Trust boundary #: 1
Untrusted party: Outsider threat, like hackers
Rationale for this boundary: (1-3 sentences) The boundary lies between users and services like public APIs outside the application and processes within the application. Everything outside the application can be called untrusted because the requests are over the public internet. The boundary helps enforce HTTPS encryption and input validation before data reaches processes inside the boundary.
Potential attack across this boundary: (1-3 sentences) Attackers could use injection attacks to compromise the registry, corrupt the model data, or get data. Additionally, since we're using the HuggingFace API and they input a URL for the download, if the attacker knows of an attack that affects the HuggingFace API, they could exploit that attack through the service to ensure we get back bad data.

Trust boundary #: 2
Untrusted party: Insider threat

Rationale for this boundary: This boundary is between our custom app code and the AWS infrastructure. The threat is essentially incompetence on our end, where any defects or exploitation could affect the rest of AWS infrastructures and throw everything off.
Potential attack across this boundary: If we don't secure this, a buggy or compromised ECS container with IAM permissions could change or delete the data in the S3 buckets or DynamoDB tables. Additionally, since we have CD from our GitHub repository, there's an extra concern that the repository or one of our accounts could be compromised and affect the ECS that way.

…

AI-Assisted Threat Modeling and Feedback

After completing your DFD(s), consult a Large Language Model (LLM) for a preliminary security analysis. The goal is to generate a broad set of security considerations and best practices for your group to critically evaluate.

1. **Request Specific Analysis:** Ask the LLM for a component-by-component security analysis. Key requests should include:
    - Specific advice on securely configuring the services you are using (e.g., "What are the security best practices for configuring an AWS S3 bucket that stores user files?" or "How should we securely configure our AWS RDS instance?").
    - General feedback on potential design weaknesses in your data flows or trust boundaries.
2. **Sub-Group Analysis:** Divide your team into two sub-groups. Each sub-group will independently query an LLM using the group's DFD. After gathering feedback:
    - As a full group, consolidate and compare the results.
    - Discuss similarities and differences in the LLM's advice.
    - Critically evaluate which recommendations are most relevant and actionable for your project.
3. **Summarize Findings:** In the area below, summarize this process. You **must** detail the specific LLM analysis or recommendations that you found most helpful or insightful. Discuss how this feedback informed your team's final security analysis and mitigation strategies.

Most useful recommendations from LLM:

- S3 Security Hardening
  - Disable public access at the bucket level.
  - Enforce bucket policies allowing access only via VPC endpoints.
  - Enable server-side encryption (SSE-S3 or SSE-KMS).
    We adopted all these recommendations.
- IAM Role Minimization
  - Separate roles for ingestion and download.
  - Explicit deny on S3 delete operations except for admin.
  - Avoid wildcard IAM actions (e.g., s3:*).
- API Gateway Rate Limiting
  - Use AWS WAF or throttling to mitigate DoS and credential-stuffing.
    We added throttling rules.
- DynamoDB Security
  - Enable point-in-time recovery (PITR).
  - Set fine-grained read/write permissions per endpoint.
- ECS Task Security
  - Enable task role instead of instance role.
  - Disable shell access and enforce read-only root filesystem.

Analysis of Threats and Mitigations

Fill out the following table for each STRIDE property (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege). For each STRIDE category, you must list specific potential threats, brainstorm multiple mitigation strategies, and then label each threat as either "Should Fix" or "Won't Fix". You must justify any "Won't Fix" decisions by explaining why the risk is low-priority, implausible, or out of scope. For every "Should Fix" threat, you must identify a single, specific mitigation strategy you plan to implement.

Note: It is acceptable if you are unable to implement *all* mitigation strategies you planned by the deadline for *this* document. However, for the final project submission, you will be required to have implemented *all* mitigation strategies for every threat you labeled as "Should Fix" in this document.

STRIDE property: Spoofing
Relevant system security properties: …
Analysis of components:

- Diagram+Component (e.g. "Diagram 1, component 3 – Database"): DFD 1 – ECS
  - Risk 1: Attacker could spoof internal service calls like API Gateway
    - Possible Mitigations: (1-2 sentences) Ensure that IAM roles are set up for authentication between services.
    - How do these address the risk: (1-2 sentences) IAM has role trust relationships that will help confirm that API Gateway is the one sending the info.
    - Suggestions for additional mitigations, if needed:
    - Decision: Fixed
    - Justification and Plan: Will ensure that we turn on per-service IAM roles with as limited privilege as possible.
- Diagram+Component: DFD 1 – Dynamo DB
  - Risk 1: Attacker could spoof the ECS and query the database
    - Possible Mitigations: (1-2 sentences) Ensure that IAM roles are set up for authentication between services, make sure we turn off any public network access to the DB.
    - How do these address the risk: (1-2 sentences) IAM has role trust relationships that will help confirm that API Gateway is the one sending the info.
    - Suggestions for additional mitigations, if needed:
    - Decision: Fixed
    - Justification and Plan: Will ensure that we turn on per-service IAM roles with as limited privileges as possible.

STRIDE property: Tampering

- Component: S3 Bucket
  - Risk: Malicious modification or deletion of model files.
    - Mitigations:
      - Enable versioning + MFA delete.
      - Restrict write permissions to ingestion role only.
      - Validate model integrity using hashes.
    - Decision: Fixed
    - Plan: Turn on versioning and restrict delete access to admin IAM role.

STRIDE property: Repudiation

…

Relevant system security properties: Non-repudiation, auditability

Analysis of components:

Diagram+Component: DFD 1 – User → API Gateway → ECS → DynamoDB/S3

Risk 1: Users can deny uploading, downloading, or modifying artifacts because there is no user authentication or audit trail.

Possible Mitigations:

- Implement user authentication (API keys or AWS Cognito) and log user identity with all operations
- Create audit trail in DynamoDB using ArtifactAuditEntry model to record user, timestamp, artifact ID, and action type
- Enable AWS CloudTrail for S3 and DynamoDB to capture service-level API calls

How do these address the risk: Authentication ties actions to identities, audit trail provides immutable records, and CloudTrail adds network-level evidence.

Decision: Fixed

Justification and Plan: Store audit entries in DynamoDB. Enable CloudTrail for S3 and DynamoDB to capture all operations with IAM principal identity and timestamps.

- **STRIDE property:** Information disclosure

  DFD 1 – API Gateway – ECS – CloudWatch Logs
  - Risk 1: Sensitive data (API keys, artifact URLs) may leak because the middleware logs full request/response bodies.
    - Possible Mitigations: Redact sensitive fields (api_key, password, token) before logging. Enable CloudWatch Logs encryption using KMS.
    - How do these address the risk: (1-2 sentences) IAM has role trust relationships that will help confirm that API Gateway is the one sending the info.
    - Decision: Fixed
    - Justification and Plan: Sanitization prevents leaking secrets, and encryption protects logs at rest. Update logging.py to redact sensitive fields; enable KMS encryption on the log group.
- DFD 1 – DynamoDB
  - Risk 1: Table data could be exposed if it's public or IAM permissions are broad.
    - Possible Mitigations: (1-2 sentences) Use high restriction on IAM access roles for the tables.
    - How do these address the risk: (1-2 sentences) Least access will ensure no accidental data leaks.
    - Suggestions for additional mitigations, if needed:
    - Decision: Fixed
    - Justification and Plan: Will ensure that we turn on per-service IAM roles with as limited privilege as possible.
- DFD 1 – API Gateway – User

- Risk 1: Error messages may reveal internal details (stack traces, file paths, DB structure).
  - Possible Mitigations: Replace detailed errors with generic user-safe messages. Keep full error details only in server-side logs.
  - How do these address the risk: (1-2 sentences) Hiding internals prevents attackers from learning about the system while still keeping logs useful for debugging.
  - Suggestion for additional mitigations, if needed: Add stronger request validation to reduce error-triggering attacks.
  - Decision: Fixed
  - Justification and Plan: Update global error handlers and route-level exceptions to return generic responses, logging the detailed ones to CloudWatch only.

STRIDE property: Denial of service

- *NB: Remember the ReDoS demonstration in class?*

…

- DFD 1 – User, API Gateway, ECS
  - Risk 1: Attackers could flood the API and cost us money
    - Possible Mitigations: (1-2 sentences) Gateway throttling by limiting requests, using WAF to filter bad traffic and using ECS autosclaing with task caps.
    - How do these address the risk: (1-2 sentences) Throttling will limit request volume, WAF is set up to block malicious patterns, and auto scaling is better at absorbing spikes.
    - Suggestions for additional mitigations, if needed:
    - Decision: Fixed
    - Justification and Plan: Will turn on throttling and auto scaling, we will work with WAF if we have time.
- DFD 1 – ECS – Dynamo
  - Risk 1: Attackers could trigger expensive scan() calls and overload DynamoDB.
    - Possible Mitigations:Replace scans with Query + GSIs on url and name_normalized. 5-second request timeouts. Add ElastiCache for hot data.
    - How do these address the risk: (1-2 sentences) GSIs avoid full-table scans, timeouts stop long requests, caching reduces load.
    - Suggestion for additional mitigations, if needed: Add strict pagination limits.
    - Decision: Fixed
    - Justification and Plan: Enforce timeouts

STRIDE property: Elevation of privilege

- *NB: If you implemented the "JSProgram" feature, tread carefully.*

…

DFD1: API Gateway – ECS

- Risk 1: An attacker could impersonate internal services (like API Gateway) and send privileged requests to ECS.
  - Possible Mitigations: Use strict IAM role trust policies so that ECS only accepts calls from API Gateway. Disable any broad or wildcard trust relationships.
  - How do these address the risk: (1-2 sentences) IAM trust rules verify the identity of the calling service, preventing unauthorized components from assuming privileged roles.
  - Suggestions for additional mitigations, if needed:
  - Decision: Fixed
  - Justification and Plan: Tighter IAM trust relationships and enforce per-service roles with least privilege.

DFD1: ECS - DynamoDB

- Risk 1:An attacker could escalate privileges by pretending to be ECS and reading/writing DynamoDB directly.
  - Possible Mitigations: Restrict DynamoDB access to the ECS task role only, and ensure the table is not publicly accessible.
  - How do these address the risk: (1-2 sentences)
  - Role-based access ensures only the ECS backend can perform table operations, blocking unauthorized callers from elevating privileges.
  - Suggestions for additional mitigations, if needed: Use VPC endpoints so DynamoDB is only reachable from internal AWS traffic.
  -
  - Decision: Fixed
  - Justification and Plan: Apply tight IAM permissions and confirm DynamoDB only accepts requests from the ECS task role

Risks resulting from component interactions

The STRIDE framework (per Microsoft) advises you to divide and conquer – analyze each component in turn. You have now done that.

Are there any instances in your system where a risk emerges from the interaction of multiple components? If you can, identify one case and describe it with reference to your DFDs. Did you find it during your STRIDE process or only just now? (1 paragraph)

A risk emerges from the interaction between the API Gateway, EC2 (backend), DynamoDB, and S3. When a user requests to enumerate or download models, the API Gateway forwards the request to EC2, which queries DynamoDB for model metadata and retrieves artifacts from S3. The risk: if an attacker compromises DynamoDB data (via IAM credentials or injection), they could inject malicious metadata that causes EC2 to fetch and serve malicious artifacts from S3, or craft metadata that triggers unintended S3 object retrieval. This risk is amplified because EC2 trusts DynamoDB metadata without re-validation, and the relationship linking logic (_link_dataset_code()) performs name-based matching that could be exploited to corrupt the artifact graph, causing EC2 to return incorrect or malicious model information through the API Gateway to the user. This wasn't identified in a STRIDE analysis, but became apparent when tracing the data flow.

Are there instances in your system where the requirements of one component (e.g., security, correctness, performance) may negatively affect the security requirements of another component or the system? If you can, identify one case and describe it. (1 paragraph)

A conflict exists between the performance requirement (avoiding re-validation on reads) and security. The DynamoDB storage layer deserializes JSON directly into Pydantic models without re-validating the artifact structure, assuming DynamoDB data is trustworthy. This optimization reduces latency but creates a security gap: if an attacker compromises DynamoDB data (via IAM credentials or injection), they can inject malformed or malicious JSON that bypasses validation. The API layer then serves this data to users without additional checks, potentially exposing corrupted artifacts, causing deserialization errors that leak information, or serving malicious content. The performance requirement (fast reads without validation overhead) conflicts with the security requirement (validate all data before serving), making the system vulnerable to data integrity attacks if the persistence layer is compromised.

Root cause analysis

Presumably (1) you did not intentionally create any security vulnerabilities, yet (2) found some through this process. Choose two interesting vulnerabilities. Describe why they happened.

Vulnerability 1:

- Our ingestion pipeline accepted arbitrary external URLs without fully validating the domain or enforcing allow listing. To mitigate, we can implement strict domain allow lists, validate the URL schema, and reject redirects to unapproved hosts.
- In early development, we prioritized functionality, assuming the users would only supply correct Hugging Face links. We did not consider how a malicious URL could cause ECS to download arbitrary files, creating a significant vulnerability.

Vulnerability 2:

- The enumerate endpoint does not have request rate limits. An attacker or faulty client could repeatedly send requests, causing ECS tasks to scale unnecessarily or saturate the S3 throughput. To mitigate, we can enable API gateway throttling and per IP rate limits.

- We initially focused on functional correctness and assumed AWS auto-scaling would cover load issues. We only recognized the risk when examining “Denial of Service” threats during STRIDE.