Hazard Analysis
Software Engineering

Team #11, OKKM Insights
Mathew Petronilho
Oleg Glotov
Kyle McMaster
Kartik Chaudhari

Table 1: Revision History

Date	**Developer(s)**	**Change**
Date1	Name(s)	Description of changes
Date2	Name(s)	Description of changes
...	...	...

Contents

[You are free to modify this template. —SS]

1 Introduction

This document presents the hazard analysis for OrbitWatch, an online application that uses an AI-powered crowdsourcing model to build labeled satellite image datasets. A hazard in the context of our application is any condition or event that could lead to undesirable outcomes, such as system failures, data corruption, data leakage, performance degradation, or security vulnerabilities. This document aims to detect, analyze, and mitigate potential hazards that are applicable to our application.

2 Scope and Purpose of Hazard Analysis

2.1 Scope

The Hazard Analysis for the OKKM Insights platform is confined to identifying and evaluating potential hazards within our internal system architecture, specifically focusing on the API backend, database, and AI model components. This analysis combines all risks associated with the application's functionality, performance, and security that could adversely affect the system's operation.

2.2 Purpose

The primary purpose of this Hazard Analysis is to systematically identify, assess, and mitigate risks that could lead to significant losses for OKKM Insights. The potential losses associated with these hazards include:

1. Monetary Losses: Prolonged platform unavailability can result in direct financial losses due to decreased service sales. Additionally, functionality disruptions may result in extra costs related to incident response and system recovery efforts.

2. Client Attrition: Downtime or unreliable performance can decrease client confidence, leading to the loss of existing clients and deterring potential new clients from engaging with our services.

3. User Attrition: A subpar user experience, characterized by inefficiencies in task labeling or system responsiveness, can drive users away from the platform. High user turnover diminishes the quality and volume of labeled data, which is essential for training accurate AI models.

By acknowledging these potential losses, the Hazard Analysis underscores the importance of proactive risk management in maintaining the platform's reliability, security, and user satisfaction.

3 System Boundaries and Components

3.1 Front End (FE)

Potential Hazards:

- **User Input Errors:** The frontend may not properly validate inputs (e.g., incorrectly labeled data or unauthorized access attempts).
- **UI Design Flaws:** Poor usability could cause users to make mistakes, such as submitting incorrect labels or missing important instructions.
- **Cross-Site Scripting (XSS):** JavaScript code that improperly checks user inputs could allow malicious scripts to be executed in the user's browser.
- **Session Hijacking:** Improper management of session tokens and cookies could allow attackers to impersonate users.

Mitigations:

- Implement thorough input validation and provide clear feedback to users.
- Ensure that critical actions like labeling are supported by warnings or confirmations to reduce errors.
- Use Content Security Policy (CSP) headers and HTML escaping techniques to prevent XSS attacks.
- Secure cookies with `HttpOnly` and `Secure` flags and employ strong session management practices.

3.2 Back End (BE)

3.2.1 Login Service

Potential Hazards:

- **Authentication Failure:** Unauthorized access could allow malicious actors to label data incorrectly or access sensitive customer information.
- **Session Hijacking:** If session management is not handled correctly, attackers could hijack valid user sessions.

Mitigations:

- Implement strong authentication mechanisms (e.g., two-factor authentication) and monitor for suspicious login activity.
- Secure cookies and session tokens, implementing timeout features.

3.2.2 Money Service

Potential Hazards:

- **Payment Failures:** Incorrect payments could cause disputes between the labelers and the platform, leading to operational delays.
- **Financial Data Breaches:** Weak encryption or improper payment processor integration could expose sensitive financial data.

Mitigations:

- Secure payment processing with encryption and multi-step verification for payout transactions.
- Implement industry-standard encryption for payment details and integrate with PCI-DSS-compliant payment gateways.

3.2.3 Task Delegation Service

Potential Hazards:

- **Task Misallocation:** If tasks are assigned to unqualified labelers, the quality of data labeling could decrease, leading to flawed training data.

Mitigations:

- Employ task delegation algorithms that consider labelers' accuracy and expertise.

3.3 Database (DB)

Potential Hazards:

- **Data Corruption or Loss:** Corrupted or missing data, especially labeled data, can introduce significant inaccuracies in model training and analysis.
- **SQL Injection:** If queries are not parameterized, attackers may manipulate query inputs, resulting in unauthorized access or data manipulation.

Mitigations:

- Implement backups and data redundancy mechanisms to prevent loss or corruption.
- Use parameterized queries or prepared statements to prevent SQL injection.
- Ensure proper data validation before storing data.

3.4 Machine Learning Task Allocation Model

Potential Hazards:

- **Model Misconfiguration:** The allocation model may incorrectly assign tasks if it misinterprets labeler performance data, leading to inefficiency and low-quality labeled datasets.
- **Bias in Task Allocation:** If the model favors certain users, this could introduce biases in the labeling process, affecting data accuracy.
- **Adversarial Attacks:** Machine learning models might be vulnerable to adversarial attacks that can skew the task allocations.

Mitigations:

- Continuously monitor and test the task allocation model's performance and adjust for fairness and accuracy.
- Implement adversarial training or model-hardening techniques to defend against attacks.

3.5 Libraries (e.g., BoTorch, TensorFlow, PyTorch)

Potential Hazards:

- **Library Bugs or Vulnerabilities:** External libraries may have bugs or security vulnerabilities that can affect system performance or introduce security risks.
- **Version Conflicts:** Dependency issues may arise when integrating BoTorch with other machine learning libraries.
- **Optimization Failures:** Incorrect hyperparameter optimization might result in poor allocation of labeling tasks.
- **Adversarial Model Attacks:** TensorFlow or PyTorch models may be susceptible to adversarial attacks, leading to incorrect classifications.

Mitigations:

- Regularly update libraries and conduct security audits to ensure all dependencies are secure.
- Use virtual environments or Docker to manage dependencies and avoid conflicts.
- Rigorously test optimization outcomes using test datasets.
- Implement adversarial defenses, such as adversarial training and data augmentation.

3.6 Python (General Back-End Processing)

Potential Hazards:

- **Performance Bottlenecks:** Python can introduce performance issues, especially in CPU-bound processes.
- **Memory Leaks:** Improper memory management can cause resource exhaustion, leading to crashes.
- **Security Vulnerabilities:** Since Python is dynamically typed, unexpected inputs could lead to runtime errors, potentially compromising data integrity.

Mitigations:

- Use Cython or external compiled languages for performance-critical tasks.
- Regularly run memory profiling tools to ensure no memory leaks.
- Implement input validation and use static analysis tools like `bandit` to enforce security best practices.

3.7 Docker (Containerization)

Potential Hazards:

- **Container Breakout:** Misconfigurations or unpatched vulnerabilities in Docker containers could allow attackers to escape the container and gain access to the host system.
- **Resource Exhaustion:** Poor resource management inside containers could lead to resource exhaustion, affecting system performance.

Mitigations:

- Use security best practices such as running containers with the least privilege, using `Seccomp` profiles, and isolating sensitive workloads.
- Set resource limits on containers to prevent exhaustion of host system resources.

4 Critical Assumptions

In developing the OKKM Insights platform, several critical assumptions have to be made about the system's functionality and reliability. These assumptions are necessary to focus the development process and mitigation efforts on areas with the highest potential hazards.

Third-Party Libraries and Maintenance

- **Assumption:** All utilized third-party libraries will remain freely available and continue to be actively maintained by their respective developers.
- **Rationale:** The platform relies on these libraries for core functionalities such as image processing, web development, and AI model training. Assuming ongoing maintenance and availability allows the team to focus on building platform-specific features without diverting resources to manage library-related issues.

Consistency of Public APIs

- **Assumption:** Public APIs integrated into the platform will maintain consistency in their logic, structure, and functionality, ensuring uninterrupted access to required data.
- **Rationale:** Reliable access to satellite imagery and other data sources through stable APIs is crucial for the platform's operations. Consistent APIs ensure that data retrieval processes remain functional without constant adjustments.

User Participation and Engagement

- **Assumption:** A sufficient number of users will engage with the platform to provide timely and accurate image labeling.
- **Rationale:** The success of the crowd-sourcing model depends on active user participation. Assuming consistent user engagement ensures that datasets can be built and updated efficiently.

Scalability of Cloud Infrastructure

- **Assumption:** The chosen cloud infrastructure will scale efficiently to handle varying loads without performance hits.
- **Rationale:** The platform's ability to handle large volumes of data and user interactions relies on scalable cloud services. Assuming scalability ensures that the system can grow with increasing demand.

Financial Stability for Compensation

- **Assumption:** Adequate financial resources will be available to compensate users for their labeling efforts.
- **Rationale:** Reliable compensation is essential for sustaining user participation and promoting high-quality data labeling.

5 Failure Mode and Effect Analysis

Design Function	Failure Modes	Effects of Failure	Causes of Failure	Detection	Recommended Action	SR	Ref
Account Creation	User already exists	User can not create an account	Email is duplicated	Compare the email entered with the user database records to see if the email is in use	Notify the user that the email is associated with another account Prompt them to give another email or sign in with the one they entered	SE2	H1-1
	Invalid input syntax and length	User can not create an account	1) Email is not valid 2) Password is not strong enough	1) Use regular expressions to detect if the string pattern is valid 2) Use regular expressions to detect if all password requirements are met	1) Notify the user that they must enter a valid email and give an example of a valid email 2) Tell the user what password requirements they have and have not satisfied	SE2,3	H1-2
Log In	Incorrect credentials entered	User can not access application	1) No account with the entered email exists 2) Password does not match records	1) Compare email entered with database records to see if account exists 2) Compare password entered with what is stored in the database for the entered email	1) Tell user account does not exist and prompt them to make one 2) Tell user password is incorrect and prompt password recovery	SE0,1	H2-1
	Excessive permissions given	Users can perform unauthorized actions	Application paths are unprotected	Check user login token each time a new page of the website is accessed	Tell the user to log in Deny access if they try to access a page they should not	SE0,1	H2-2
Labeling Satallite Images	Internet connection is lost	Users can not submit labeled images or navigate the website	Internet connection is weak or power is lost	Device shows no internet connection	Any labeled photos or created projects that have already been submitted have been saved Progress is resumed when connection is re-established	SE4	H3-1
	Application is closed	Same as H3-1	Power outage or misclick	Application is no longer running on the users device	Any labeled photos or created projects that have already been submitted have been saved Progress is resumed on log in	SE4	H3-2
	Unlabelled data is submitted	Bad data is added to the dataset	Misclick	On submission, application checks that there are as many labels as requested by the job	Reject a submission if no labeling was done	SE5	H3-3
	Mass labeling done too quickly	Bad data is added and reward system is abused	Bots have been deployed to make quick labels	User is submitting data at an unreasonable speed	Implement a submission cool down to prevent bot submissions Reward system is based on accuracy	SE5	H3-4
Backend Server and API Requests	Server crashes	All services provided by the server are down	Software error on server side	Error found in logs	Monitor errors in logs Notify users that the server is down	SE6	H4-1
	API is not responding	All services provided by an API do not work	API service provider is down or overwhelmed	Response from the API has an error code	Retry all API requests after a specific amount of time Monitor errors in logs	SE6	H4-2

Data Storage	User account is compromised	User info is exposed and they will be dissatisfied with the application	Lack of encryption and protection of sensitive infomation	User notifies the team of lost reward balance or lost account access	Ensure user passwords are encrpyted when stored Ensure financial transactions are secure Password reset occurs through a trusted source such as email	SE8,9	H5-1
	Duplicate entry occurs	Data inconsistency, unneccesary storage usage, and slower query performance	Lack of constraints/validation	Check the database entries	Ensure the database has unique keys Set up a duplicate key procedure on the database	SE7	H5-2
	Database is compromised	Data inconsistency, malicous entries, and data leaks	SQL injection	Check database entries	Use parameterized queries Avoid dynamic SQL strings	SE10	H5-3

6 Safety and Security Requirements

A comprehensive list of requirements can be found in the SRS.

NFR-SE4

- **Description:** The application shall prohibit new user actions when user is disconnected from the system.
- **Rationale:** The system should avoid batch updates upon reconnect as to prevent malicious users creating a false package of updates.
- **Fit Criterion:** Upon user disconnect and reconnect, the system will not accept any updates from the user not submitted before the disconnect occured.

NFR-SE6

- **Description:** The system shall, during system failure, notify users trying to make a request of the system failure.
- **Rationale:** The system should provide good warning messages to users to prevent additional frustration.
- **Fit Criterion:** When services are unavailable, the system shall provide an error message to the user making the request of the unavailable service.

NFR-SE7

- **Description:** The system shall reject duplicate entries in database.
- **Rationale:** Each entry to the database should be a unique data piece. If two entries are identical, they must differ by at least the creation time.

- **Fit Criterion:** No two entries in the database shall exist together for longer than 24 hours from their introduction into the database.

7 Roadmap

Primary Requirements The first priority for the system is to ensure integrity in the labels collected. This means we will first prioritize requirements related to the user authentication process (**NFR-SE0, NFR-SE1**), data integrity (**NFR-SE5, NFR-SE6, NFR-SE7**), and user privacy(**NFR-SE8, NFR-SE9**).

Secondary Requirements Additionally, the system must also be prepared for malicious actions from users. This is out of scope for completion during the capstone course, but must be implemented before exposing the system to a large pool of users. These requirements are related to malicious inputs (**NFR-SE10**) and potentially harmful inputs (**NFR-SE2, NFR-SE3**). In the event of a system failure, we would also like to ensure that error messages are helpful to users (**NFR-SE4**).

Appendix — Reflection

[Not required for CAS 741 —SS]

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

Kartik

1. What went well while writing this deliverable?

   During this deliverable, I had the opportunity to research topics related to an app's safety and the threats it may be subjected to. Previously, when writing about hazards, I was only aware of some basic risks that I had heard about since childhood. However, investigating each topic related to our SRS made me think in-depth about other potential hazards. I would say the aspect of research based on existing SRS went really well, as I learned a lot of new things. Although this deliverable was relatively shorter than the previous ones, I developed the habit of starting early. In the back of my mind, I knew what to do and what was coming early on when this assignment was released. Another positive aspect was starting this assignment earlier and establishing a flow of ideas in my head.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   One of the pain points was identifying specific issues related to our project. The libraries we are going to use aren't definitive yet, and since they may change in the future, I had to anticipate potential hazards concerning future libraries we might use. I did write specific risks related to a few of them, but generalizing these risks was challenging, especially predicting the problems we might encounter, which is a necessary skill to develop.

3. Which of your listed risks had your team thought of before this deliverable, and which did you think of while doing this deliverable? For the latter ones (the ones you thought of while doing the Hazard Analysis), how did they come about?

   I was already aware of SQL injection and API vulnerabilities, as I learned about them during my internship and even had a chance to address them when we encountered a situation at my workplace. For the frontend, I understood how deprecated libraries can cause significant issues and how important it is to use stable versions during the development phase. While working on this deliverable, I learned more about Cross-Site Scripting (XSS), Session Hijacking, and how unpatched vulnerabilities in Docker could allow attackers to gain access to our system. Additionally, learning about the mitigations was very useful, as I had to delve deeper to understand how I could implement them for our project. Overall, I found the process to be quite solid, and as I mentioned, I learned a lot during this journey, including watching in-depth videos on hazards related to Docker containers.

4. Other than the risk of physical harm (some projects may not have any appreciable risks of this form), list at least two other types of risk in software products. Why are they important to consider?

   One potential risk related to software products is Server-Side Request Forgery (SSRF). In this case, an attacker could make connections to internal-only services within the organization's infrastructure, allowing them to make requests to unintended locations. Another potential risk is a more generic one—Insecure Design. This broad category of risks relates to how a system behaves when something goes wrong. When an attacker tries to compromise our application, we need a set of automated instructions to prevent the attacker from accessing other data or to stop further misuse. Having strong security policies and sufficient validation for user inputs are a few examples of measures to avoid such problems while also improving designs in areas prone to failure.

8 References

- OWASP. Cross-Site Scripting (XSS). Available at: https://owasp.org/www-community/attacks/xss/.
- The SSL Store. The Ultimate Guide to Session Hijacking (aka Cookie Hijacking). Available at: https://www.thesslstore.com/blog/the-ultimate-guide-to-session-hijacking-aka-cookie-hijacking/.
- Mozilla Developer Network. Content Security Policy (CSP). Available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.
- Neontri. PCI DSS Compliance. Available at: https://neontri.com/blog/pci-dss-compliance/.
- Imperva. Session Hijacking. Available at: https://www.imperva.com/learn/application-security/session-hijacking/.
- Label Your Data. Bias in Machine Learning. Available at: https://labelyourdata.com/articles/bias-in-machine-learning.

- Reversing Labs. How to Harden ML Models Against Adversarial Attacks. Available at: https://www.reversinglabs.com/blog/how-to-harden-ml-models-against-adversarial-attacks.
- Hyperopt GitHub Issues. Hyperopt Version Conflicts. Available at: https://github.com/hyperopt/hyperopt/issues/642.
- HiddenLayer. What's in the Box? Understanding Adversarial Attacks on ML Models. Available at: https://hiddenlayer.com/research/whats-in-the-box/.
- Medium. Python Security Best Practices for Writing Secure Code. Available at: https://medium.com/@VAISHAK_CP/python-security-best-practices-for-writing-secure-code-a6a9130e3748.
- Docker Documentation. Seccomp Security Profiles. Available at: https://docs.docker.com/engine/security/seccomp/.