An Exploratory Study of Security Vulnerabilities in Machine Learning Deployment Projects

Akond Rahman* Anthony Skjellum† Yue Zhang‡
* Auburn University, AL, USA
† Tennessee Tech University, TN, USA
Email: *akond@auburn.edu †askjellum@tntech.edu ‡yzz0229@auburn.edu

Abstract—**Machine learning (ML) deployment projects are used by practitioners to automatically deploy ML models. While ML deployment projects aid practitioners, security vulnerabilities in these projects can make ML deployment infrastructure susceptible to security attacks. A systematic characterization of vulnerabilities can aid in identifying activities to secure ML deployment projects used by practitioners. We conduct an empirical study with 149 vulnerabilities mined from 12 open source ML deployment projects to characterize vulnerabilities in ML deployment projects. From our empirical study, we (i) find 68 of the 149 vulnerabilities are critically or highly severe; (ii) derive 10 consequences of vulnerabilities, e.g., unauthorized access to trigger ML deployments; and (iii) observe established quality assurance activities, such as code review to be used in the ML deployment projects. We conclude our paper by providing a set of recommendations for practitioners and researchers. Dataset used for our paper is available online.**

Index Terms—**devops, mlops, security, vulnerabilities**

I. Introduction

Machine learning (ML) has become a ubiquitous technology that directly impacts multiple aspects of modern society. Such ubiquitous usage necessitates efficient deployment of models that are trained using ML algorithms. Yet, practitioners face challenges when deploying ML models to end-users. According one survey, 87% of constructed ML models are not deployed [1]. Furthermore, ML-specific factors, such as model security pose challenges for deploying ML models in cloud-based or mobile-based infrastructure. In order to alleviate these challenges, experts have recommended usage of automation practices, such as ML deployment [13]. ML deployment is the practice of automatically deploying ML models to end-users using tools, such as Apache Airflow [9], [13]. Usage of ML deployment tools have yielded benefits for organizations. For example, practitioners from Adidas have reported that usage of Apache Airflow helped them to setup and run 1,000 ML deployments to serve their ML models [1].

Despite being beneficial for organizations, ML deployment infrastructure can be susceptible to security attacks as vulnerabilities reside in ML deployment projects. Let us consider the example in Figure 1 that shows a vulnerability reported for Apache Airflow, an open source project used for ML deployment [1]. The vulnerability [2] is reported as a critical vulnerability with a GitHub Security Advisory (GHSA) score of 9.3 out of 10. The vulnerability occurred because of providing unnecessary privileges for all users by default using `default: "airflow.api.auth.backend.default"`. This vulnerability could allow malicious entities to gain unauthorized access to a ML deployment infrastructure that is setup using Apache Airflow. The vulnerability shown in Figure 1 highlights the importance of securing ML deployment infrastructure by systematically investigating the nature of vulnerabilities in ML deployment projects. Such investigation can be beneficial for (i) maintainers to gain insights on how to securely develop ML deployment projects; and (ii) researchers to be informed on future research directions.

```
version_added: ~
type: string
example: ~
default: "airflow.api.auth.backend.default"
default: "airflow.api.auth.backend.deny_all"
```

Fig. 1: Code snippet showcasing a security vulnerability observed in 'Apache Airflow'.

Accordingly, we answer the following research questions:

- **RQ1**: *What is the severity of security vulnerabilities that occur in machine learning (ML) deployment projects? What are the consequences of vulnerabilities that occur in ML deployment projects?*
- **RQ2**: *What quality assurance activities are used by ML deployment projects?*

We conduct an empirical study with 149 vulnerabilities obtained for 12 open source ML deployment projects. We use open coding [14] to derive consequences of vulnerabilities that occur in ML deployment projects. We also inspect the content of each project repository to determine what quality assurance activities are used by projects.

Contributions: We list our contributions as follows:

[1]https://d2iq.com/blog/why-87-of-ai-ml-projects-never-make-it-into-production-and-how-to-fix-it

[2]https://github.com/advisories/GHSA-hhx9-p69v-cx2j

```
from airflow import DAG
...
# Define the DAG
dag = DAG(
    'sample_deployment_pipeline',
    ...
)
# Define the task dependencies
preprocess_task = PythonOperator(
...
)
train_task = PythonOperator(
...
)
evaluate_task = PythonOperator(
...
)
preprocess_task >> train_task >> evaluate_task
```

Fig. 2: Code snippet demonstrating how a ML deployment pipeline can be created with Apache Airflow.

- An empirical evaluation of vulnerability severity for open source ML deployment projects; and
- A derived list of consequences for vulnerabilities that occur in open source ML deployment projects.

II. Background and Related Work

A. Background

ML deployment is the practice of deploying ML models to end-users in an automated fashion [9], [13]. Deployment of ML models can occur using on-prem servers, cloud-based infrastructure, or edge devices [11]. Deployment of ML models have certain properties that differentiate them from deployment of traditional software: (i) *data distribution shifts*: the phenomenon of when there is change for the underlying distribution of the data using which the model was trained [9]; (ii) *model drifting*: the phenomenon of a models' predictive power decreasing over time because of changes in the domain in which the model is being applied to, as well as changes in correlation between variables [13]; and (iii) *model packaging*: the practice of packaging model artifacts, such as annotations, dependencies, configuration files, and metadata into a single entity for deployment and reuse [13].

Figure 2 presents an example of how ML model deployment can be conducted with Apache Airflow. We observe two constructs called `DAG` and `PythonOperator` to be used, which is respectively used to define the relationships between tasks, and define a task that needs to be executed as part of the deployment. We observe three tasks to be specified namely using `PythonOperators`, namely `preprocess_task`, `train_task`, and `evaluate_task`.

B. Related Work

Our paper is closely related with prior research that has investigated security weaknesses in ML projects. Xiao et al. [19] studied 15 security vulnerabilities in deep learning libraries and derived 3 threats. Spring et al. [17] conducted thought experiments to identify differences between the ML and Common Vulnerability Exposure (CVE) communities. Papernot et al. [12] studied and synthesized security attacks that can be conducted against ML algorithms. Their study was used by Bhuiyan and Rahman [4] who derived a taxonomy of coding patterns that need to be used to perform attack-related forensics for ML projects. Harzevili et al. [7] conducted an empirical study with 683 vulnerabilities, and derived vulnerability categories, root causes, and symptoms. A similar study was conducted by Bhuiyan et al. [3], who studied 3,964 security vulnerabilities from 278 OSS projects that use supervised learning algorithms. Researchers have also studied techniques that can be used for detecting security vulnerabilities in ML-related projects. Harzevili et al. in separate publications evaluated the techniques of static analysis [6] and fuzzing [16] to identify security vulnerabilities in ML-related projects.

While we observe existing research to systematically study security vulnerabilities in ML libraries and projects that use ML, there is a lack of understanding on how frequently security vulnerabilities appear in ML deployment projects. We address this research gap in our paper.

III. Methodology

We provide the methodology to answer the research questions:

A. Methodology to Answer RQ1

1) Tool Identification: We conduct a grey literature review with Internet artifacts to identify the projects that practitioners use for ML deployment. Using the Google search engine in incognito mode, we search with the string 'machine learning deployment projects'. Using the collected top 100 search results, we apply the following criteria: (i) *Criterion-1*: The project must be related to ML deployment; *Criterion-2*: The project must be open source; and *Criterion-3*: The project must have vulnerability data recorded by GHSA. Applying our filtering criteria, we identify 12 projects that satisfy the three criteria.

2) Vulnerability Data Collection: We use GHSA to identify vulnerabilities that are reported for the 12 projects. We use GHSA as it provides a curated list of vulnerabilities for all projects that are hosted publicly available on GitHub.

3) Answer to RQ1: We report the frequency of vulnerabilities by reporting the severity of the vulnerabilities. We also apply a qualitative analysis technique called open coding [14] to derive the consequences of the vulnerabilities. The first and last author apply open coding individually. The Cohen's Kappa is 0.63, indicating 'substantial' agreement [10]. In the case of disagreements, the first author's decision is final.

B. Methodology to Answer RQ2

1) Identification of Quality Assurance Activities: Quality assurance activities, such as security application security testing tools (SASTs) are advocated for secure development of software projects [18]. Quantifying quality assurance activity

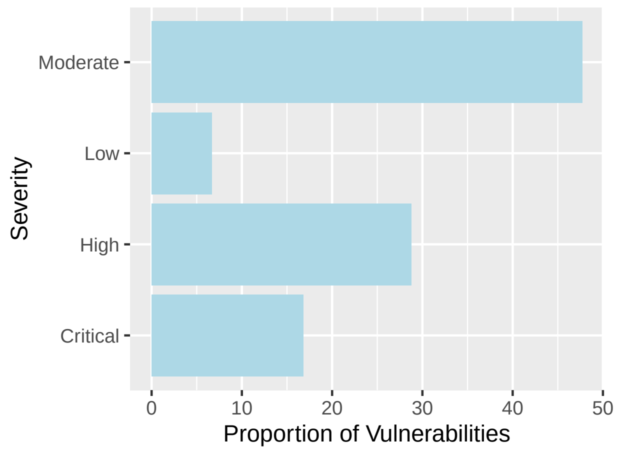

Fig. 3: Answer to RQ1: Severity of studied vulnerabilities.

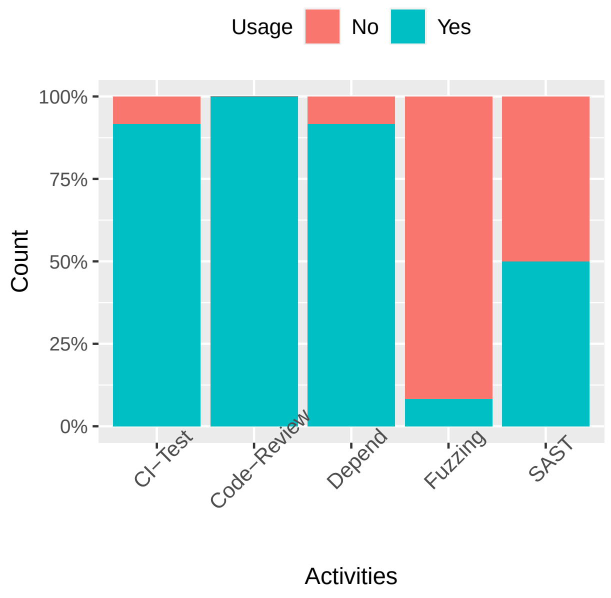

Fig. 4: Answer to RQ2: Frequency of quality assurance activities used in ML deployment projects.

usage in ML deployment projects can provide an understanding of the activities that are used in ML deployment projects. We inspect the content of the repository for each project to determine usage for of the following activities:

Continuous Integration (CI)-based Testing: The activity of using a CI tool, such as TravisCI to test code changes before merging into the project. The activity of testing code changes is a pivotal activity in CI, and is also reported to aid increasing quality of the project [15].

Code Review: The activity of reviewing source code. We use this activity as prior research [5] shows usage of code review to detect and mitigate vulnerabilities.

Dependency Upgrade: The activity of using automated tools for dependency upgrades. We use this activity as usage of automated dependency upgrade tools can aid in repairing vulnerable software dependencies [8].

Fuzzing: The activity of using invalid data as inputs to computer programs in an automated manner to identify bugs [2]. We use this activity as usage of fuzzing is recommended by experts for secure software development [2].

SAST: The usage of SASTs to identify bugs. SAST is also recommended by experts for secure software development [18].

2) Answer to RQ2: We answer RQ2, by reporting the proportion of the projects that use each of the five activities.

IV. Results

A. Answer to RQ1

In all, we identify 149 vulnerabilities. Of the 149 vulnerabilities, 25 and 43 are respectively ‘critical’ and ‘high’ vulnerabilities. The most frequent category is ‘moderate’ with 47.6% of the 149 vulnerabilities. All relevant data related to severity is available in Figure 3.

We identify 10 consequences, amongst which the most frequent consequence is unauthorized access. The names and definitions of 10 consequences is reported in Table I, which is sorted based on ‘Prop’ . The ‘Prop (%)’ column reports the proportion of vulnerabilities mapped to each consequence.

B. Answer to RQ2

We answer RQ2 by using Figure 4 where the x-axis lists the activities, and y-axis shows the proportion of projects that use a certain activity. We observe code review to be the most frequently used activity, whereas, fuzzing is the least frequently used activity.

V. Discussion

A. Implications

1) Implications for Practitioners: One of the implications of our findings is to advocate the usage of fuzzing for ML deployment projects. Fuzzing has been shown to be an effective technique for finding security vulnerabilities over the years [2], and we posit that this activity could be helpful identifying latent vulnerabilities in ML deployment projects as well. Also, as shown in Figure 4, we observe majority of the projects to use recommended quality assurance activities, such as SAST and code review. Practitioners who are new to ML deployment projects can use this finding to secure ML deployment projects.

2) Implications for Researchers: Our findings lay the groundwork for multiple possible directions of future research. *First*, researchers can invest in creating a comprehensive dataset of vulnerabilities unique to ML deployment, by curating multiple data sources. *Second*, using the curated dataset researchers can evaluate the strengths and weaknesses of vulnerability identification techniques for ML deployment projects. *Third*, upon evaluating the vulnerability identification techniques, researchers can address the limitations on improving existing

TABLE I: Answer to RQ1: Consequences of vulnerabilities in ML deployment projects with definitions.

Consequence	**Definition**	**Prop (%)**
Unauthorized access	The consequence of an unauthorized entity gaining access to an ML deployment infrastructure.	62.4
Injection of malicious content	The consequence when a malicious content can be injected.	10.7
Exposure of information	The consequence that leads to exposure of sensitive information, such as passwords.	9.4
Incorrect redirection	The consequence when a user is redirected to the incorrect destination.	4.7
Denial of service	The consequence that makes the ML deployment project unavailable.	4.0
Authentication error	The consequence that leads to incorrect authentication of users.	2.7
Unexpired sessions	The consequence when a user session remains unexpired allowing it to be used by a malicious entity.	2.7
Insecure secrets	The consequence that makes secret data, such as authentication tokens ineffective.	2.0
Incorrect provisioning	The consequence that leads to incorrect provisioning of ML deployment infrastructure.	0.7
Incorrect rendering	The consequence when incorrect content is rendered on the user interface.	0.7

vulnerability identification tools. This activity could identify new vulnerabilities and security attacks that have not been reported in prior research.

B. Threats to Validity

The limitations of our paper are:

Conclusion Validity: Our paper is susceptible to a lack of conclusion validity as we use GHSA as the only data source. While this ensures more reliability in our vulnerability set, it also reduces the diversity in the collected set of vulnerabilities.

External Validity: Our results are limited to ML deployment projects that are open source. Our findings may not generalize to projects that are proprietary, such as Amazon Sagemaker.

VI. Conclusion

Despite being pivotal for serving ML-based services to end-users, ML deployment projects can be susceptible to security vulnerabilities, causing serious consequences. We have conducted an empirical study with 149 vulnerabilities, where we observe 45.6% of the studied vulnerabilities to be critically or highly severe. We also observe majority of the studied projects to use established quality assurance activities, such as code review. Based on our findings, we recommend researchers to evaluate existing vulnerability identification techniques to advance the knowledge of secure ML deployment.

Acknowledgments

We thank the PASER group at Auburn University for their valuable feedback. This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2312321.

References

[1] Apache, "Apache airflow," https://airflow.apache.org/, 2024, [Online; accessed 29-Oct-2024].

[2] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, "Finding software vulnerabilities by smart fuzzing," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 2011, pp. 427–430.

[3] F. A. Bhuiyan, S. Prowell, H. Shahriar, F. Wu, and A. Rahman, "Shifting left for machine learning: An empirical study of security weaknesses in supervised learning-based projects," in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2022.

[4] F. A. Bhuiyan and A. Rahman, "Log-related coding patterns to conduct postmortems of attacks in supervised learning-based projects," *ACM Trans. Priv. Secur.*, vol. 26, no. 2, apr 2023.

[5] L. Braz, C. Aeberhard, G. Çalikli, and A. Bacchelli, "Less is more: supporting developers in vulnerability detection during code review," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1317–1329.

[6] N. S. Harzevili, J. Shin, J. Wang, S. Wang, and N. Nagappan, "Automatic static vulnerability detection for machine learning libraries: Are we there yet?" in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, 2023, pp. 795–806.

[7] ——, "Characterizing and understanding software security vulnerabilities in machine learning libraries," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, 2023, pp. 27–38.

[8] R. He, H. He, Y. Zhang, and M. Zhou, "Automating dependency updates in practice: An exploratory study on github dependabot," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4004–4022, 2023.

[9] C. Huyen, *Designing machine learning systems*. " O'Reilly Media, Inc.", 2022.

[10] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[11] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: a survey of case studies," *ACM computing surveys*, vol. 55, no. 6, pp. 1–29, 2022.

[12] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *2018 IEEE European Symposium on Security and Privacy (EuroSP)*, 2018, pp. 399–414.

[13] E. Raj, *Engineering MLOps: Rapidly build, test, and manage production-ready machine learning life cycles at scale*. Packt Publishing Ltd, 2021.

[14] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.

[15] A. Sharif, D. Marijan, and M. Liaaen, "Deeporder: Deep learning for test case prioritization in continuous integration testing," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 525–534.

[16] N. Shiri Harzevili, M. M. Mohajer, M. Wei, H. V. Pham, and S. Wang, "History-driven fuzzing for deep learning libraries," *ACM Trans. Softw. Eng. Methodol.*, aug 2024, just Accepted.

[17] J. M. Spring, A. Galyardt, A. D. Householder, and N. VanHoudnos, "On managing vulnerabilities in ai/ml systems," in *Proceedings of the New Security Paradigms Workshop 2020*, ser. NSPW '20. New York, NY, USA: Association for Computing Machinery, 2021, p. 111–126.

[18] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way (Paperback) (Addison-Wesley Professional Computing Series)*, 1st ed. Addison-Wesley Professional, 2011.

[19] Q. Xiao, K. Li, D. Zhang, and W. Xu, "Security risks in deep learning implementations," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018.