Risk Classification of Docker Container Images Using Machine Learning and Vulnerability Metrics

Santosh Ugale
Department of Computer Engineering, MET's Institute of Engineering, Nashik 422003, Maharashtra, Affiliated to Savitribai Phule Pune University (SPPU), Maharashtra, India
ugalesantosh5@gmail.com (corresponding author)

Amol Potgantwar
Department of Computer Engineering, Sandip Institute of Technology and Research Center, Nashik 422213, Maharashtra, Affiliated to Savitribai Phule Pune University (SPPU), Maharashtra, India
amol.potgantwar@sitrc.org

Received: 8 June 2025 | Revised: 7 October 2025 and 11 October 2025 | Accepted: 13 October 2025

 DOI: https://doi.org/10.48084/etasr.12627

ABSTRACT

With the rapid adoption of containerized applications, ensuring the security of container images has become a critical concern. Traditional image scanning tools provide a list of vulnerabilities but lack automated risk classification mechanisms that aid in proactive mitigation. This research presents a Machine Learning (ML)-based approach to classify container images into High-Risk and Low-Risk categories using metadata and vulnerability scan results. The dataset was generated by scanning widely used Docker images with Trivy, capturing attributes such as image size, number of packages, file count, executables, and CVE severity levels. Two XGBoost-based classification models were developed. The first model used raw features, achieving an accuracy of 90.91%. Employing the same datasets, the second model achieved 100% accuracy using engineering features, specifically Vuln_Score (Critical + High vulnerabilities) and Pkg_per_MB (package density). The results show that adding domain-specific features improves risk detection accuracy and provides a scalable way to automate security assessments in CI/CD pipelines. This study proposes an effective method for classifying container images and detecting security flaws for different containerized platforms.

Keywords-cloud security; containerization; DevSecOps; vulnerability assessment; docker

I. INTRODUCTION

The wide use of containerization platforms and applications provides easy scalability and efficiency, and has accelerated software deployment. However, these advancements have led to new security risks. While containers are lightweight and portable, they might have security vulnerabilities that affect the entire application stack. Firewalls and signature-based detection techniques are examples of traditional security solutions that frequently fail to detect advanced risks in container environments. With a 99% F1 score and 100% AUC, authors in [1] proposed a random forest algorithm for identifying infected and malicious Docker images. Research has shown that containerized platforms require enhanced security measures. In [2], it was demonstrated that the number of vulnerabilities increased with time, even with frequent provider updates. Debian-based images exhibited more vulnerabilities than other Linux distributions.

Authors in [3] investigated ML approaches for container security, highlighting their potential for intrusion and anomaly detection within 5G networks. They found that these container images contain various vulnerabilities. Software updates could remove these vulnerabilities, while removing unused packages was also effective [4]. Similarly, authors in [5] leveraged Convolutional Neural Networks (CNNs) to detect malware in Docker containers by analyzing their file systems. Authors in [6] discussed an automated security framework for containerized applications and highlighted the key challenges and solutions in the Docker container platforms. With their comprehensive taxonomy of container vulnerabilities, authors in [7] offered significant insights into major attack methods and mitigation techniques.

Authors in [8] proposed SHIL, a self-supervised framework, which identifies and improves the online detection of security attacks in containerized applications. Authors in [9] demonstrated the great potential of deep learning in security applications by using CNNs to detect hidden threats in Docker

containers automatically. Container platforms need to keep up to date with the latest versions and consider security practices to reduce the security surface [10]. However, automated security, vulnerability monitoring, and alerting facilitate breaking silos between dev, ops, and sec teams by offering access to essential security metrics [11].

An ML-based Intrusion Detection and Prevention System (IDPS) was presented in [12] to defend cloud networks against brute force and DDoS attacks. A tutorial-based framework [13] explored challenges and evaluation approaches for deploying microservice architectures in DevOps settings. Current security concerns, future risks, and open challenges in containerized environments were analyzed in [14].

A testing methodology was presented in [15] to detect software security vulnerabilities by assessing the changes in system posture during agile development. A comparative analysis of leading cloud providers was highlighted in [16], examining differences in data protection, threat response, and compliance services. A real-world case study [17] demonstrated the use of open-source tools for automating security in industrial software engineering workflows. A cloning-based security strategy [18] was proposed for securing Virtual Machines (VMs) in cloud environments. Instead of running applications directly on the main VM, cloned instances were created, and the application was deployed on these clones.

Trivy [19] was employed as the primary vulnerability scanner due to its ease of use, broad ecosystem support, and ability to detect CVEs in both operating system packages and application dependencies. Grype [20], another open-source tool, offers deep integration with Software Bill of Materials (SBOMs) and is particularly suitable for automated CI/CD environments and air-gapped infrastructure.

Traditional scanning tools, such as Trivy and Grype, effectively identify known vulnerabilities (CVEs) but lack the intelligence to assess the overall risk levels. They also tend to generate verbose reports, making it difficult for DevSecOps teams to prioritize and act upon threats promptly.

Docker Hub provides native image vulnerability scanning through Snyk [21], enabling developers to assess the risks associated with published containers. While useful for surface-level inspection, it lacks contextual risk prioritization, a gap addressed by the ML classifier proposed in the current study.

This study explores ML, specifically the XGBoost Classifier, to classify Docker images into High-Risk or Low-Risk based on features extracted from vulnerability scans. The current work begins with a basic model using five operational metrics, followed by a refined version that introduces two engineered features:

- Vuln_Score: The combined count of critical and high-severity CVEs.
- Pkg_per_MB: The ratio of total packages to image size.

These two model developments demonstrate the impact of feature engineering on classification performance. The current study's findings validate that the Model-2 performs better than the Model-1, achieving 100% accuracy and highlighting the potential of ML for identifying and automating container risk assessments. Container images are a real attack vector rather than a hypothetical one. When implementing such workloads, organizations should implement suitable preventative and proactive security measures that can mitigate the threats that target cloud platforms [22].

In this context, the present study introduces an ML-based framework for assessing the security risk of Docker images. By analyzing metadata and vulnerability scan results, the study proposes categorizing images into High-Risk and Low-Risk groups and enabling preventative security steps. The proposed method uses the XGBoost algorithm, a popular and effective ML algorithm well-known for its stability and effectiveness when working with structured data, which makes it the ideal match for the research.

Given that containers provide isolated, portable, and lightweight execution environments, they have entirely changed how software is deployed. Concerns over container image security are growing as more businesses use containerized microservices. The vulnerabilities embedded in base images or third-party packages can propagate across production environments, posing a serious threat to enterprise infrastructure.

II. METHODOLOGY

The present work investigated a VM, with the software and hardware specifications presented in Table I.

TABLE I. HARDWARE AND SOFTWARE DETAILS

Hardware/ software	Specifications/version details
CPU	Single Core with four logical CPUs (Intel)
Operating system	Ubuntu 24.04 LTS
Memory	8 GiB RAM
Storage	180 GB HDD
Docker	Version: 26.1.3
Trivy	Version 0.67.0
Grype	Version 0.100.0
Python	Python 3.7.12

A. Data Collection

A sample dataset was prepared by selecting 51 publicly available Docker images to ensure that the study reflects realistic, widely-used software environments. Images from several categories, including web servers (httpd 2.4.65, nginx 1.25), databases (MySQL 8.1, MongoDB 7.0, PostgreSQL 15.2), language runtimes (Node 18, Ruby 3.1, Python 3.11), and DevOps tools (Redis 7.0, RabbitMQ 3.11, Telegraf 1.35.3) were used [23]. Since these images are freely available, frequently updated, and widely utilized in both development and production environments, they were considered suitable for analysis. The images were retrieved on an Ubuntu 24.04 LTS environment with Docker 26.1.3 installed, using the standard Docker pull command (e.g., `#docker pull httpd`). [24].

Pulled images are stored under the Docker local storage path /var/lib/docker. Inside this directory, overlay2 contains unpacked image layers. The image folder contains metadata (Manifest and configuration), and the container folder contains runtime container files.

B. Dataset Description

All downloaded images vary in size (measured in MB), consist of multiple layers, and include various binary packages and executables. The Docker image command displays the list of images downloaded on the Ubuntu system along with their repository name, tag, image ID, creation date, and image size. Table II presents the information provided by the Docker images. The objective is to study and analyze vulnerabilities in Docker images and classification, using different image types [25].

`[root@localhost ~]# docker images`: This Docker image command provides the Docker image repository name and image size. The study uses the Grype and Trivy tools to obtain more detailed information about the pulled images, including the number of packages, total files, and executables inside the images. To extract and identify the vulnerabilities in container images, Trivy and Grype were installed on the Ubuntu 24.04 LTS server, and Docker container images were scanned using commands 1 and 2.

TABLE II. DOCKER CONTAINER IMAGES DETAILS

Repository	Tag	Image Id	Created	Size
nginx	latest	4cad75abc83d	6 months ago	192MB
httpd	latest	10fd72f437c4	7 months ago	148MB

To scan a container image for vulnerabilities, the following command was used:

```
trivy image Container_image_name
[root@localhost ~]# trivy image nginx:1.25 | grep "Total"                    (1)
2025-10-04T09:59:06-04:00       INFO [vuln] Vulnerability scanning is enabled
2025-10-04T09:59:06-04:00       INFO [secret] Secret scanning is enabled
2025-10-04T09:59:06-04:00 INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2025-10-04T09:59:06-04:00 INFO    [secret]
2025-10-04T09:59:06-04:00       INFO Detected OS family="debian" version="12.5"
2025-10-04T09:59:06-04:00 INFO    [debian] Detecting vulnerabilities... os_version="12" pkg_num=149
2025-10-05T09:59:06-04:00 INFO    Number of language-specific files num=0
2025-10-04T09:59:06-04:00 WARN    Using severities from other vendors for some vulnerabilities.
Total: 274 (UNKNOWN: 2, LOW: 119, MEDIUM: 92, HIGH: 48, CRITICAL: 13)
[root@localhost ~]# grype httpd:2.4.65 (2)
✓ Pulled image
✓ Loaded image  httpd:2.4.65
✓Parsed image
sha256:2416cb32cb59e9ac3de2bf99ab60a1e2f889917bb8687af48e3781ae2af41776
✓Cataloged contents
9e36877050f027646ed13538c6ed7287f5255cbc2bc227f64e27102fb411b1d5
 ├── ✓ Packages      [113 packages]
 ├── ✓ Executables   [837 executables]
 ├── ✓ File metadata [2,714 locations]
 └── ✓ File digests  [2,714 files]
✓ Scanned for vulnerabilities  [84 vulnerability matches]
 ├── by severity: 0 critical, 6 high, 10 medium, 5 low, 63 negligible
```

Docker builds images layer by layer. The number of layers in a Docker image is an important metric because each layer influences the performance, efficiency, and security of the container. The former has both advantages and disadvantages: more layers enable fine-grained caching and faster builds, but excessive layers can slow down image pulls. Also, analyzing layers helps identify which specific layer introduces vulnerabilities. With the use of the Docker inspect command and an additional parameter, the number of layers can be displayed.

```
[root@localhost ~]# docker inspect nginx --format='{{ len .RootFS.Layers }}'
```

In the above example, layers were checked for the image name nginx. Similarly, the number of layers was calculated for all images considered for this study [23]. The number of layers obtained from the Docker inspect command, along with the total packages, files, executables, and critical, high, medium, and low vulnerabilities obtained from the Grype scan, are summarized in Table III.

The scan results provided the following quantitative attributes for each image:

- Image size (in MB), total layers, total packages installed, and number of files.
- Number of executable files, CVE severity count: critical, high, medium, and low.

C. Model Design and Deployment

The data displayed in Table I were converted to CSV format in a cleaned and normalized way, including removing the unnecessary columns and unnamed headers. The cleaned data were used for further training and testing of the model. This study introduces and tests two models: one with default dataset features and the second with two additional engineered features.

- Model 1: XGBoost with default 5 raw features, which include image size (MB), total layers, total packages installed, total files, and executables.
- Model 2: XGBoost with additional engineered features (Vuln_Score and Pkg_Per_MB). The total features include image size (MB), total layers, total packages installed, total files, executables, Vuln_Score, and Pkg_per_MB. The Vuln_Score and Pkg_per_MB were calculated using:

- Vuln_Score: The sum of critical and high-severity CVEs

$$Vuln\_Score = Critical + High \quad (1)$$

- Pkg_Per_MB: Density of installed packages to the image size in MB.

$$Pkg\_Per\_MB = \frac{Total\ Packages\ installed}{Image\ Size\ (MB)} \quad (2)$$

Both models were trained utilizing an 80/20 train-test split with stratified sampling to preserve class distribution. The models were evaluated using accuracy, precision, Recall, F1-score, and related metrics.

TABLE III. IMAGE SCAN RESULTS FOR IMAGES FROM THE DATASET REPORTED IN [23, 25]

Sr. No	Image name	Image size (MB)	Total layers	Total packages	Files	Executables	Critical	High	Medium-	High
1	almalinux:9	189MB	1	155	651	5245	0	2	8	0
2	fedora:39	177MB	1	147	586	4512	0	0	0	0
3	busybox:1.36	4.42MB	1	1	11	1	0	0	4	2
4	caddy:2.8.4	49.2MB	5	144	26	231	3	10	20	6
5	cassandra:4.0	358MB	10	231	945	5706	8	39	51	56
6	couchdb:3.4	271MB	11	259	814	5252	2	14	28	20
7	curlimages/curl:8.16.0	29.9MB	3	38	68	147	0	2	4	6
8	debian:11	124MB	1	96	702	5398	1	9	18	7
9	elasticsearch:9.0.5	1.37GB	10	804	430	2036	1	6	23	39
10	alpine:3.18	7.36MB	1	15	17	79	0	0	0	6
11	gcc:11	1.23GB	8	414	1363	19066	70	772	1332	138
12	golang:1.21	814MB	7	227	1068	13385	32	605	1186	55
13	haproxy:2.8	100MB	6	82	659	2600	1	3	8	5
14	httpd:2.4.65	117MB	6	113	837	2714	0	6	10	5
15	influxdb:2.5	232MB	10	358	750	3584	37	157	168	36
16	joomla:4.4	775MB	23	399	1277	11565	13	43	71	29
17	mariadb:10.6	305MB	8	152	926	6018	8	32	1167	50
18	mediawiki:1.41	1.04GB	21	370	1322	10064	24	373	643	61
19	memcached:1.6	85MB	6	83	666	2444	1	2	5	5
20	mongo:7.0	834MB	8	370	775	3976	8	40	39	39
21	mysql:8.1	574MB	10	142	415	16830	8	58	85	17
22	neo4j:4.4	542MB	6	348	811	5581	3	196	611	21
23	nextcloud:25	1.07GB	21	436	1302	10731	69	929	1646	95
24	nginx:1.25	188MB	7	150	842	3710	14	60	77	19
25	node:18	1.09GB	8	619	1331	19682	39	228	377	90
26	openjdk:21	504MB	3	114	375	2768	0	32	59	19
27	opensuse/leap:15.6	115MB	1	140	555	2229	0	0	0	0
28	oraclelinux:9	239MB	1	195	889	6065	0	1	8	0
29	perl:5.38	1.08GB	7	458	1371	20913	34	121	92	83
30	photon:4.0	42.2MB	1	36	164	471	0	0	0	0
31	php:8.1	524MB	10	180	840	8113	2	15	34	7
32	postgres:15.2	379MB	13	149	932	7709	18	128	106	34
33	python:3.11	1.1GB	7	481	1417	21563	34	128	101	83
34	quay.io/prometheus/prometheus:v2.50.0	254MB	12	337	4	3	8	20	44	4
35	rabbitmq:3.11	217MB	11	107	778	3823	2	5	82	53
36	redis:7.0	109MB	8	93	704	2995	10	51	51	12
37	redmine:5.0	630MB	13	407	1301	9026	37	127	124	41
38	rockylinux/rockylinux:8.6	196MB	1	152	656	6825	0	45	251	244
39	ruby:3.1	987MB	7	500	1434	19564	40	274	471	114
40	rust:1.60	1.3GB	6	411	1324	18806	125	1582	2625	204
41	solr:9.5	580MB	11	665	822	5189	1	26	99	63
42	swift:5.9	2.56GB	4	241	1096	10359	0	4	1774	180
43	telegraf:1.35.3	520MB	6	656	865	8021	2	17	32	20
44	tomcat:10.1	469MB	9	189	894	4305	0	0	113	64
45	ubuntu:22.04	77.9MB	1	101	733	2291	0	0	8	32
46	vault:1.10.0	199MB	5	335	31	985	17	68	68	10
47	wordpress:6.8.2-php8.3-apache	734MB	24	268	1297	10421	9	36	64	20
48	znc:1.8	475MB	6	108	388	10365	14	54	56	8
49	hylang:1.1.0	131MB	5	108	796	3206	1	7	17	8
50	django:1.10.4	436MB	6	186	1108	9847	10	330	933	412
51	Owncloud 10.0.10	10.0.10	21	618	292	8352	1095	200	675	599

III. RESULTS

The performance metrics for both models were analyzed and reported. The first model used raw scan metrics, such as image size (MB), total layers, total packages installed, total files, and executables, as input features. Table IV provides an overview of Model 1 metrics, and Table V outlines Model 2 metrics. Figures 1-4 present the results and evaluation of Model 1.

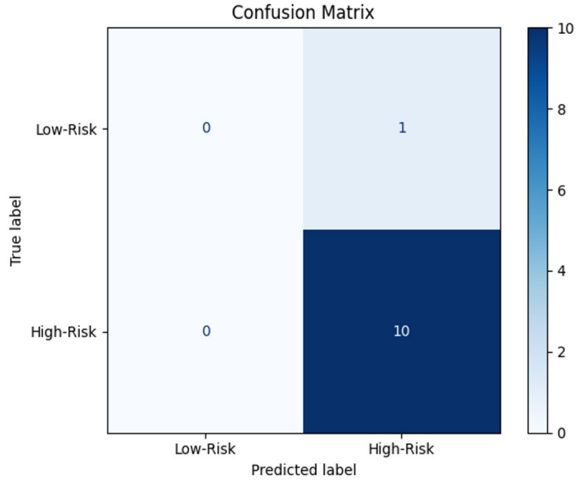

Fig. 1. Confusion matrix for Model 1.

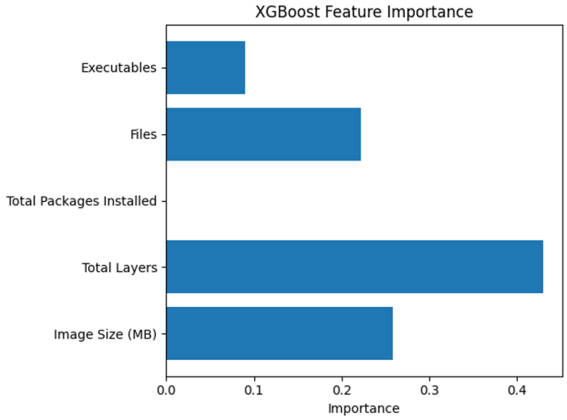

Fig. 2. XGBoost feature importance for Model 1.

TABLE IV. MODEL 1 METRICS

	Precision	Recall	F1-score	Support
0	0.00	0.00	0.00	1
1	0.91	1.00	0.95	10
Accuracy			0.91	11
Macro avg	0.45	0.50	0.48	11
Weighted avg	0.83	0.91	0.87	11

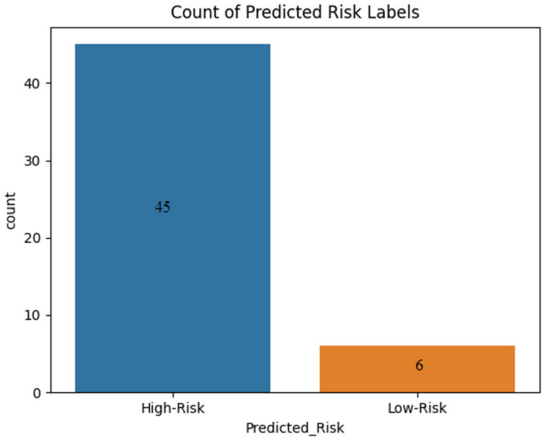

Fig. 3. Count of predicted risk labels for Model 1.

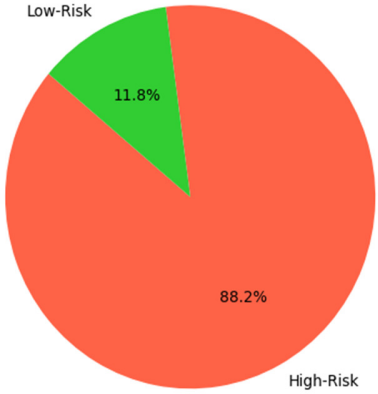

Fig. 4. Risk prediction distribution for Model 1.

For the XGBoost classifier with Vuln_Score and Pkg_per_MB, Figures 5-8 illustrate the results and evaluation of Model 2.

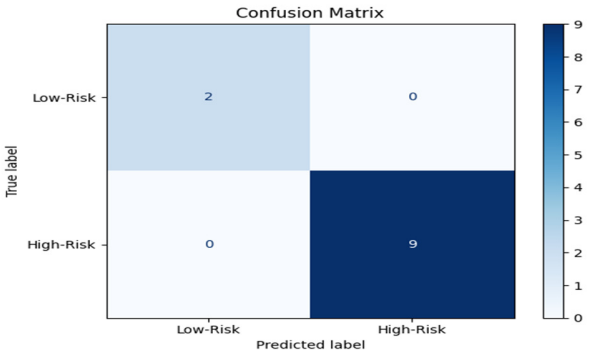

Fig. 5. Confusion matrix for Model 2.

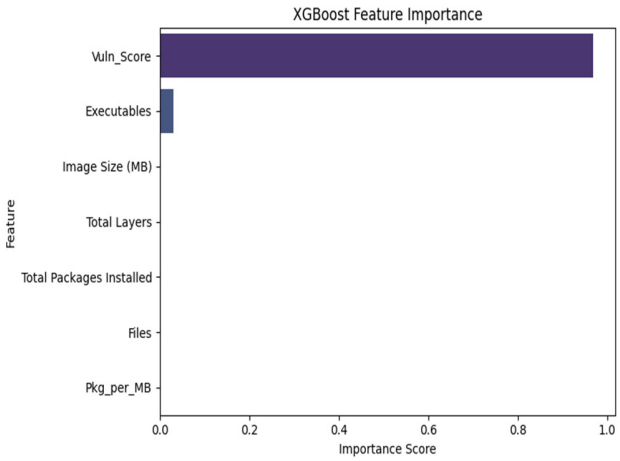

Fig. 6. XGBoost feature importance for Model 2.

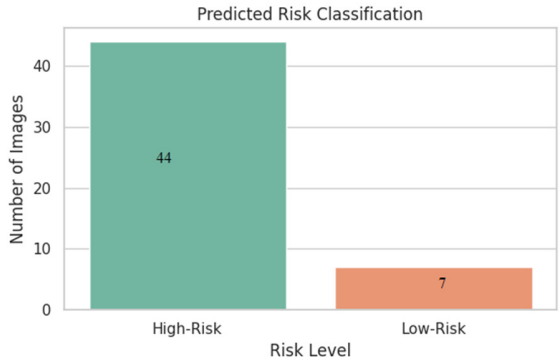

Fig. 7. Predicted risk classification for Model 2.

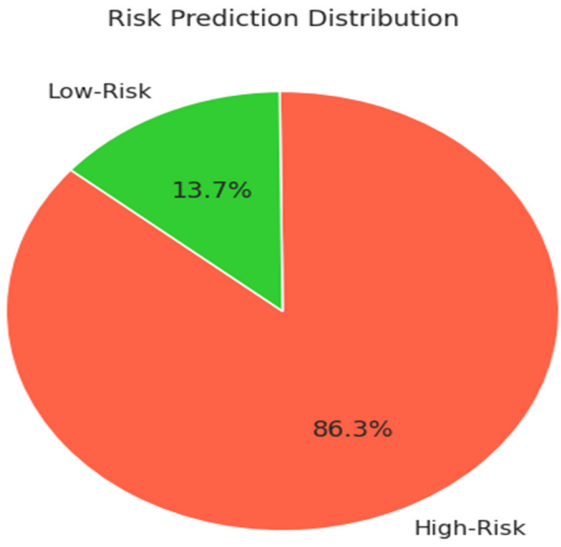

Fig. 8. Risk prediction distribution for Model 2.

The second model, an XGBoost classifier with Vuln_Score, incorporated feature engineering by adding Vuln_Score (Critical + High CVEs) and Pkg_Per_MB (package density).

TABLE V. MODEL 2 METRICS

	Precision	**Recall**	**F1-score**	**Support**
0	1.00	1.00	1.00	2
1	1.00	1.00	1.00	9
Accuracy			1.00	11
Macro avg	1.00	1.00	1.00	11
Weighted avg	1.00	1.00	1.00	11

IV. DISCUSSION

Model 1 used raw scan metrics, such as image size (MB), total layers, total packages installed, total files, and executables, as input features, and it was trained using an XGBoost classifier, achieving 90.91% accuracy. Model 2, an XGBoost classifier with Vuln_Score, incorporated feature engineering by adding Vuln_Score (Critical + High CVEs) and Pkg_Per_MB (package density). Model 2 achieved 100% accuracy on the same evaluation framework with enriched features.

TABLE I. MODEL 1 AND MODEL 2 COMPARISON

Aspect	**Model 1 XGBoost Classifier**	**Model 2 XGBoost Classifier with Vuln_Score**
Features	Image size (MB), total layers, total packages installed, total files, executables	Image size (MB), total layers, total packages installed, total files, executables, Vuln_Score, Pkg_per_MB
Engineered features	None	*Vuln_Score* = *Critical* + *High*, Pkg_per_MB
Accuracy	90.91% (on the dataset in Table I)	100% (on the dataset in Table I)
Model	XGBoost classifier	XGBoost classifier
Train-test split	80/20	80/20 with stratify = *y*
Evaluation metrics	Accuracy, precision, F1-score, recall	Accuracy, precision, F1-score, recall

The change from raw operational metrics to domain-aware features improved performance and enhanced interpretability, as confirmed by the feature importance analysis and confusion matrix review. Table VI shows that Model 1 achieved moderate accuracy but misclassified a few low-risk images as high risk. Moreover, Model 2 results were improved with domain-specific engineered features. These points highlight the value of feature engineering in enhancing the sensitivity and specificity of ML-based classification in security tasks.

V. CONCLUSION

This research presents a practical and scalable Machine Learning (ML) approach to automate the risk classification of Docker container images. By leveraging the metadata of container images and the vulnerability metrics obtained from tools like Trivy, the study developed and compared two models based on the XGBoost classifier algorithm.

The first model was trained on basic scan attributes, such as image size (MB), total layers, total packages installed, total files, and executables, and achieved a modest accuracy of 90.91%. Feature engineering was introduced to improve performance by adding two new attributes: Vuln_Score and

Pkg_per_MB. The second model, enhanced with these features, achieved 100% accuracy on the same test dataset and separated High-Risk and Low-Risk images. Using domain-specific features significantly improved the classifier's precision and recall.

Adding domain-specific knowledge (Vuln_Score, Pkg_per_MB) helped the model detect patterns, such as more vulnerabilities with high risk. The engineered features reduced the burden on the model to learn implicit relationships and helped balance the stratified split to ensure fair evaluation by maintaining class ratios across training and test sets.

ACKNOWLEDGMENT

The authors thank Ashok Pomnar at ESDS Software Solution for his valuable input and support during this research. The authors also extend their gratitude to the MET Institute of Engineering Research Center, Nashik, and ESDS Software Solution for providing resources and support.

DATA AVAILABILITY STATEMENT

The Dataset used in this study is publicly available at https://www.kaggle.com/datasets/santoshugale/container-image-scan-results.

REFERENCES

[1] M. Aldiabat, Q. M. Yaseen, and Q. A. Ein, "An Efficient Random Forest Classifier for Detecting Malicious Docker Images in Docker Hub Repository," *IEEE Access*, pp. 1–1, 2024, https://doi.org/10.1109/ACCESS.2024.3506663.

[2] A. Mills, J. White, and P. Legg, "Longitudinal Risk-Based Security Assessment of Docker Software Container Images," *Computers & Security*, vol. 135, Dec. 2023, Art. no. 103478, https://doi.org/10.1016/j.cose.2023.103478.

[3] B. Kaur, M. Dugré, A. Hanna, and T. Glatard, "An Analysis of Security Vulnerabilities in Container Images for Scientific Data Analysis," *GigaScience*, vol. 10, no. 6, June 2021, Art. no. giab025, https://doi.org/10.1093/gigascience/giab025.

[4] O. Tunde-Onadele, Y. Lin, X. Gu, J. He, and H. Latapie, "Self-Supervised Machine Learning Framework for Online Container Security Attack Detection," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 19, no. 3, pp. 1–28, Sept. 2024, https://doi.org/10.1145/3665795.

[5] J. Diaz, J. E. Perez, M. A. Lopez-Pena, G. A. Mena, and A. Yague, "Self-Service Cybersecurity Monitoring as Enabler for DevSecOps," *IEEE Access*, vol. 7, pp. 100283–100295, 2019, https://doi.org/10.1109/ACCESS.2019.2930000.

[6] M. Nadeem, A. Arshad, S. Riaz, S. S. Band, and A. Mosavi, "Intercept the Cloud Network from Brute Force and DDoS Attacks via Intrusion Detection and Prevention System," *IEEE Access*, vol. 9, pp. 152300–152309, 2021, https://doi.org/10.1109/ACCESS.2021.3126535.

[7] S. Sultan, I. Ahmad, and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019, https://doi.org/10.1109/ACCESS.2019.2911732.

[8] I. T. Aktolga, E. S. Kuru, Y. Sever, and P. Angin, "AI-Driven Container Security Approaches for 5G and Beyond: A survey," *ITU Journal on Future and Evolving Technologies*, vol. 4, no. 2, pp. 364–382, June 2023, https://doi.org/10.52953/ZRCK3746.

[9] A. Nousias *et al.*, "Malware Detection in Docker Containers: An Image is Worth a Thousand Logs," in *ICC 2025 - IEEE International Conference on Communications*, Montreal, QC, Canada, June 2025, pp. 6401–6407, https://doi.org/10.1109/ICC52391.2025.11161263.

[10] N. Jaccard, T. W. Rogers, E. J. Morton, and L. D. Griffin, "Automated Detection of Smuggled High-Risk Security Threats using Deep Learning," in *7th International Conference on Imaging for Crime Detection and Prevention (ICDP 2016)*, Madrid, Spain, 2016, Art. no. 11 (4.) -11 (4.), https://doi.org/10.1049/ic.2016.0079.

[11] O. Jarkas, R. Ko, N. Dong, and R. Mahmud, "A Container Security Survey: Exploits, Attacks, and Defenses," *ACM Computing Surveys*, vol. 57, no. 7, pp. 1–36, July 2025, https://doi.org/10.1145/3715001.

[12] Md. S. Islam Shamim, F. Ahamed Bhuiyan, and A. Rahman, "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices," in *2020 IEEE Secure Development (SecDev)*, Atlanta, GA, USA, Sept. 2020, pp. 58–64, https://doi.org/10.1109/SecDev45635.2020.00025.

[13] A. Avritzer, "Challenges and Approaches for the Assessment of Micro-Service Architecture Deployment Alternatives in DevOps: A tutorial presented at ICSA 2020," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Salvador, Brazil, Mar. 2020, pp. 1–2, https://doi.org/10.1109/ICSA-C50368.2020.00007.

[14] B. Arnold and Y. Qu, "Detecting Software Security Vulnerability during an Agile Development by Testing the Changes to the Security Posture of Software Systems," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, Dec. 2020, pp. 1743–1748, https://doi.org/10.1109/CSCI51800.2020.00323.

[15] A. Guptha, H. Murali, and S. T, "A Comparative Analysis of Security Services in Major Cloud Service Providers," in *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, May 2021, pp. 129–136, https://doi.org/10.1109/ICICCS51141.2021.9432189.

[16] F. Angermeir, M. Voggenreiter, F. Moyon, and D. Mendez, "Enterprise-Driven Open Source Software: A Case Study on Security Automation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Madrid, Spain, May 2021, pp. 278–287, https://doi.org/10.1109/ICSE-SEIP52600.2021.00037.

[17] S. Ugale and A. Potgantwar, "Container Security in Cloud Environments: A Comprehensive Analysis and Future Directions for DevSecOps," in *RAiSE-2023*, Dec. 2023, Art. no. 57, https://doi.org/10.3390/engproc2023059057.

[18] N. K. A. Nemirajaiah and C. K. Raju, "Securing Virtual Machines using Cloning in Cloud Services," *Engineering, Technology & Applied Science Research*, vol. 15, no. 2, pp. 20770–20775, Apr. 2025, https://doi.org/10.48084/etasr.9391.

[19] Aqua Security. "Trivy - A Simple and Comprehensive Vulnerability Scanner for Containers and other Artifacts." GitHub Repository. https://github.com/aquasecurity/trivy.

[20] *Grype - Vulnerability Scanner for Container Images and Filesystems*, v0.102.0, GitHub Repository, 2025 [Online]. Available: https://github.com/anchore/grype.

[21] Docker Inc. "Docker Hub - Manage Image Vulnerability Scanning." Docker Documentation, 2025, https://docs.docker.com/docker-hub/repos/manage/vulnerability-scanning/.

[22] Sysdig. "Analysis of Supply Chain Attacks Through Public Docker Images." Sysdig Blog, 2025, https://sysdig.com/blog/analysis-of-supply-chain-attacks-through-public-docker-images/.

[23] S. Ugale, "Container image dataset." Zenodo, Oct. 10, 2025, https://doi.org/10.5281/ZENODO.17316682.

[24] "Docker Hardened Images - Secure & Compliant." Hub, 2025, [Online]. Available: https://hub.docker.com/.

[25] Santosh Ugale, "Container Image Scan Results." Kaggle Dataset, Oct. 2025, [Online]. Available: https://www.kaggle.com/datasets/santoshugale/container-image-scan-results.