Technical Artefacts

**Jay Dilipbhai Yadav**

**(12226402)**

Table of Contents

[Model Integration 3](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425101)

[Random Forest Model (best\_model.pkl) 3](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425102)

[TF-IDF Vectorizer (vectorizer.pkl) 3](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425103)

[Label Encoder (label\_encoder.pkl) 4](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425104)

[Workflow of Integration 4](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425105)

[Advantages of Pre-trained Component Integration 4](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425106)

[AWS Deployment 5](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425107)

[EC2 Instances 5](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425108)

[1. Key Instance Details 6](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425109)

[2. Additional Tabs and Features 6](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425110)

[3. Significance in the Project 6](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425111)

[Challenges and Mitigations 7](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425112)

[Justification of Selected Methods 8](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425113)

[Conclusion 10](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425114)

[References 11](file:///C:\Users\jayya\Downloads\Jay%20Dilipbhai%20Yadav.docx#_Toc187425115)

**Model Integration**

The following three pre-trained components are integrated into Flask app utilising joblib for accurate predictions.

A computer code with text

Description automatically generated with medium confidence

*Figure 1 Model Integration*

The Flask app uses joblib to blend three pre-trained components for reliable web vulnerability predictions. These components have been trained and serialised into.pkl files for optimised loading and reuse in the Flask app. Prediction pipeline components include Random Forest Model, TF-IDF Vectorizer, and Label Encoder (Sangra et al., 2024).

**Random Forest Model (best\_model.pkl)**

Flask uses **Random Forest Model prediction,** and this text-based vulnerability classifier was trained. Random Forest was chosen because:

* **Robustness**: Multiple decision trees decrease overfitting and increase accuracy.
* **High Accuracy**: Random Forest's ensemble nature ensures reliable predictions even with noisy or convoluted data.
* **Feature Importance Analysis**: The most important features for predictions are assessed by Random Forest, enhancing interpretability (Ao et al., 2019).

The pre-trained Random Forest model, best\_loaded.pkl, is serialised using joblib in the Flask app. Without retraining, the app may use the model immediately, reducing computing overhead. The model predicts vulnerability using pre-processed and vectorised user-submitted text data patterns.

**TF-IDF Vectorizer (vectorizer.pkl)**

As per (Liu et al., 2018) Machine learning models use numerical features from raw text extracted by the TF-IDF Vectorizer. It generates TF-IDF scores that represent word importance relative to the dataset from input text. Some important TF-IDF steps:

* **Term Frequency (TF)**: Measures document word app frequency.
* **Inverse Document Frequency (IDF)**: Weights unusual words across publications to reduce common terms.

The pre-trained vectorizer, Vectorizer.pkl, ensures input text is converted to model training data. Disparities between training and testing data formats can harm model performance, therefore alignment is crucial for accurate predictions. The Flask app loads the vectorizer from joblib to preprocess text data provided by users.

**Label Encoder (label\_encoder.pkl)**

Label Encoder turns Random Forest model predictions into human-readable vulnerability classes. SQL Injection, Cross-Site Scripting, and other vulnerabilities are labelled numerically during model training. As an example:

* "SQL Injection" → 0
* "Cross-Site Scripting" → 1

After prediction, pre-trained label\_encoder.pkl labels numeric output. This step ensures users obtain clear results. These findings are dynamically shown on the admin page of the Flask app, increasing user experience.

**Workflow of Integration**

1. **Model Loading**:
   * All three pre-trained components (best\_model.pkl, vectorizer.pkl, label\_encoder.pkl) are loaded into the Flask app during initialisation using Joblib. Predictions are made more quickly.
2. **Prediction Process**:
   * The web interface accepts text.
   * TF-IDF vectorizer is used to vectorise pre-processed input.
   * The Random Forest model predicts numeric labels from vectors.
   * Numeric labels become human-readable vulnerability types with the Label Encoder.
3. **Dynamic Result Display**:
   * The admin page dynamically shows the decoded result, providing users with prompt feedback (Singh and Paul, 2020).

**Advantages of Pre-trained Component Integration**

1. **Efficiency**:
   * The computational effort and delay of model retraining upon app start are reduced by pre-training and serialisation.
2. **Consistency**:
   * The TF-IDF vectorizer and Label Encoder match model training data input and output.
3. **Scalability**:
   * Modular pre-trained components allow quick updates or replacement without major Flask application changes (Singla and Sandarsh Chavalmane, 2023).

In conclusion, joblib integration of the pre-trained Random Forest Model, TF-IDF Vectorizer, and Label Encoder boosts Flask application efficiency, accuracy, and usability. These components work together to give users reliable and intelligible predictions, making the system a good tool for identifying web vulnerabilities.

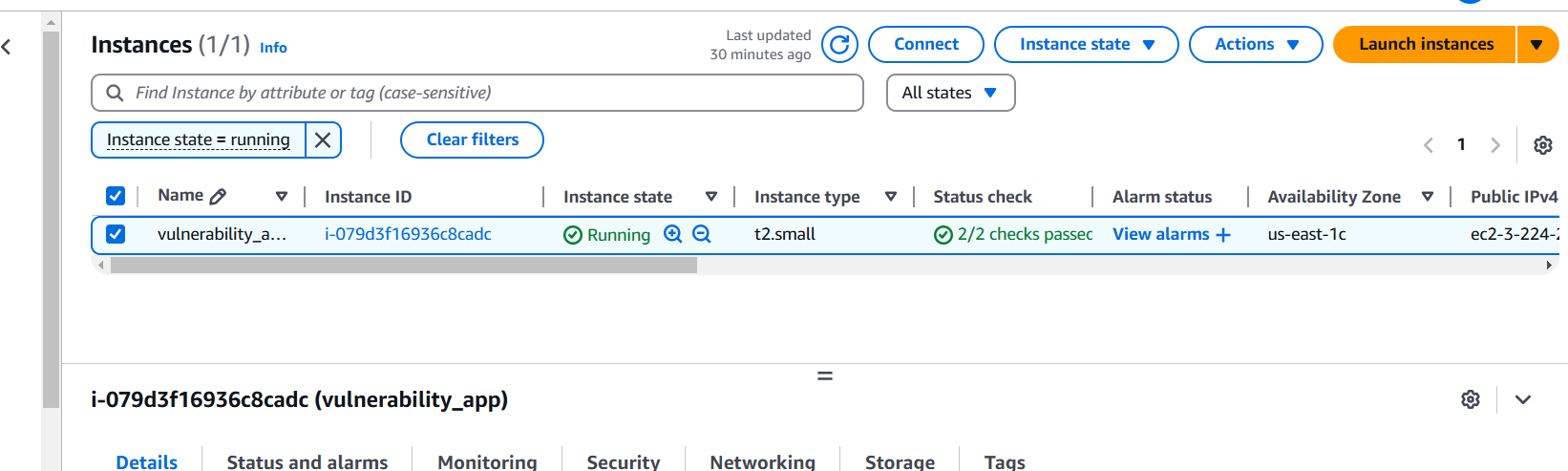
**AWS Deployment**

**EC2 Instances**

For app web apps and backend models, EC2 Instances offer a scalable and reliable computational environment. According to (Goh, Ho and Abas, 2022) Amazon EC2 delivers instances, virtual machines, for cloud app development. This project hosts the Flask-based web application and machine learning backend on EC2 instances, ensuring smooth deployment and interaction between users and the prediction system.

EC2 instances may adjust computation, memory, storage, and network capacity for applications. An EC2 instance hosts the Flask-built web application, making it live. Random Forest model, TF-IDF vectorizer, and label encoder are pre-trained backend model components integrated in the same instance. The application may process user inputs, predict, and display results in real time using this setup.

Scalability is EC2's main benefit. Traffic and workload determine instance upgrade or auto-scaling. The web application is protected by EC2 security groups. By hosting the web application and backend model on an EC2 instance, the project delivers a cost-effective and robust app deployment approach. The pay-as-you-go pricing model ensures efficient resource consumption, making EC2 ideal for app hosting cloud-based machine learning applications. The system can give precise and reliable predictions thanks to this integration's excellent availability and performance.



*Figure 2 AWS EC2 instance management console*

The AWS EC2 (Elastic Compute Cloud) management console displays an instance's setup and status. The vulnerability detection system's web application and backend model are hosted by ID (i-079d3f16936c8cadc) and named vulnerability. Flask-based application and machine learning backend run on this EC2 instance at deployment.

**Key Instance Details**

1. **Instance Type**:
   * The instance type **t2.small** balances compute, memory, and networking. This instance type is cost-effective for lightweight applications and development environments.
2. **Instance State**:
   * The instance is now in the **running** state, indicating it is ready to process requests.
3. **Status Checks**:
   * The instance passed its **2/2 checks**, indicating a proper system and network setup. The checks ensure instance reliability and workload readiness (Amazon Web Services (AWS) (n.d.-b) .
4. **Public IPv4 Address**:
   * For external hosting application access, the instance has a **public IPv4 address (3.224.206.40).** This address is used by users of the web app to communicate (Amazon Web Services (AWS) (n.d.-a).
5. **Availability Zone**:
   * For high availability and minimal latency, the instance is in **us-east-1c**. It ensures app accessibility and performance.

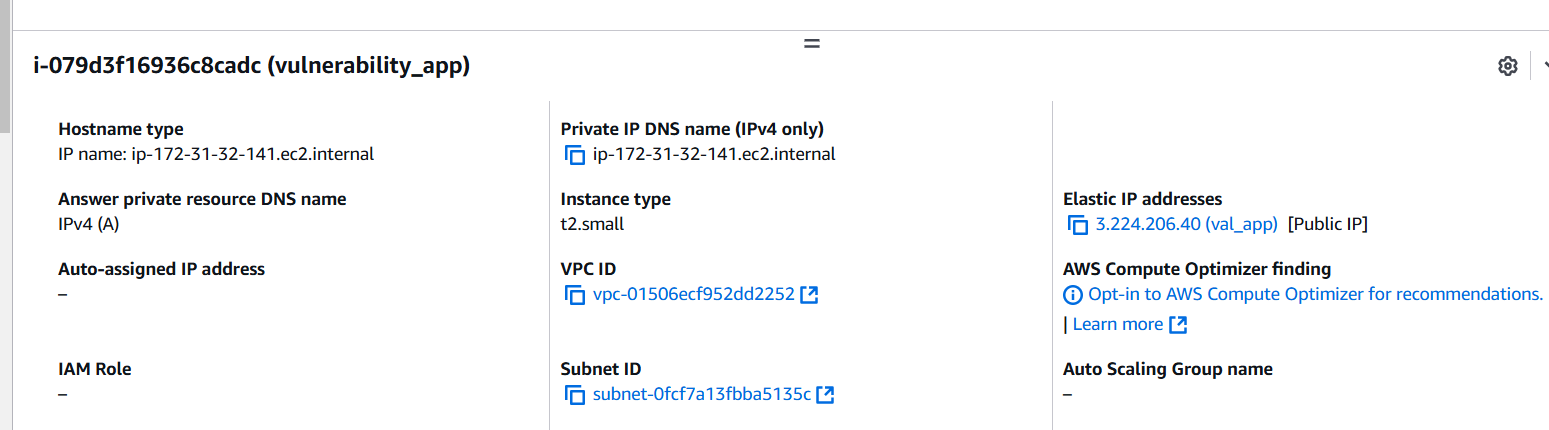
**Additional Tabs and Features**

* The console has tabs for additional information:
  + **Monitoring**: Monitor CPU, network, and disc performance in real time.
  + **Security**: Security configurations and groups control inbound and outbound traffic as firewalls.
  + **Networking**: Instance VPC, subnets, and Elastic IPs are displayed (Sampaio and Barbosa, 2020).

**Significance in the Project**

The EC2 instance hosts the Flask web app and machine learning backend. It streamlines application interactions and predicts vulnerabilities in real time using input data. The public IPv4 address ensures accessibility, and the t2.small type gives enough resources for the application without overspending.

The reliable and performance-optimized instance is us-east-1c and status checks. This setup ensures efficient operation and good vulnerability detection service delivery, making EC2 important to the project ecosystem.



*Figure 3 AWS EC2 instance with ID*

Visualisation of an AWS EC2 instance with ID i-079d3f16936c8cadc, marked vulnerability\_app, which hosts the application and its backend services. T2.small is a cheap instance with balanced compute, memory, and network capacity. For lightweight workloads, this instance type is ideal for hosting a Flask-based web app with a machine learning backend.

For instance, private and public networks are configured. VPC internal communication uses its Private IP DNS name (ip-172-31-32-141.ec2.internal). The association of a Public Elastic IP address (3.224.206.40) ensures consistent external connectivity, making the application accessible over the internet. Elastic IPs provide service availability even after instance reboots.

The instance controls routing tables, security groups, and subnets in a secure, isolated VPC (vpc-01506ecf952dd2252). For security and control, subnet-0fcf7a13fbba5135c divides the network. In the future, an IAM Role can be added to the instance to secure programmatic access to AWS services like S3 for data storage or CloudWatch Monitoring (Amazon Web Services (AWS) (n.d.-c)) .

The instance lacks an Auto Scaling Group, indicating that dynamic scaling is not configured. This setup is great for apps with known traffic patterns. The AWS Compute Optimiser shows resource optimisation for cost-efficiency and performance. App hosting for vulnerability detection application deployment is secure and reliable with this arrangement.

**Challenges and Mitigations**

|  |  |  |
| --- | --- | --- |
| **Challenge** | **Details** | **Mitigation** |
| **Model Integration with Flask** | Compatibility issues between pre-trained components (Random Forest, TF-IDF, Label Encoder) and Flask integration. | Used Joblib for efficient serialization/deserialization of models and ensured a streamlined prediction workflow for seamless component interaction. |
| **Preprocessing Pipeline Alignment** | Discrepancies between training data format and incoming user data affected prediction accuracy. | Applied the same preprocessing steps (TF-IDF vectorization and label encoding) for training and real-time predictions to maintain consistency. |
| **Real-Time Prediction Latency** | Delays due to model size and processing overhead during real-time predictions. | Selected a lightweight Random Forest model optimized for smaller datasets to ensure quicker predictions without sacrificing accuracy. |
| **AWS EC2 Configuration** | Balancing cost, performance, and scalability when selecting an instance type for hosting the application. | Used t2.small instance type for cost-effective hosting with adequate resources. Associated Elastic IP for stable connectivity and ensured optimal configuration for performance. |
| **Public Accessibility and Security** | Maintaining public accessibility while ensuring secure communication between users and the application. | Configured Virtual Private Cloud (VPC) with isolated subnets and security groups. Used Elastic IP for stable connectivity and HTTPS for secure data transmission. |
| **Deployment in a Scalable Environment** | Preparing for traffic spikes while keeping costs manageable. | Leveraged AWS EC2's scalability and pay-as-you-go model for resource adjustments. Deployed CloudWatch for performance monitoring and scaling optimization. |
| **Ensuring Accuracy in Real-World Scenarios** | Difficulty handling edge cases such as incomplete or ambiguous input data. | Enhanced the preprocessing pipeline to handle noisy and incomplete inputs. Added error handling mechanisms to provide informative feedback for invalid inputs. |
| **Disaster Recovery and Backup** | Protecting critical data and ensuring service continuity in case of failures. | Implemented AWS Backup for automated, policy-driven backups. Configured disaster recovery plans with cross-region backups to enhance resilience. |

**Justification of Selected Methods**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component** | **Selected Method** | **Why This Method Was Selected** | **Alternative Methods** | **Why Other Methods Were Not Selected** |
| **Machine Learning Model** | **Random Forest** | - High accuracy and robustness against overfitting. - Works well with smaller datasets. - Feature importance analysis enhances interpretability. - Suitable for multi-class classification. | Neural Networks Support Vector Machine (SVM) | - Neural Networks require high computational resources and are complex to train. - SVM struggles with scalability for multi-class problems. |
| **Feature Extraction** | **TF-IDF Vectorizer** | - Converts textual data into numerical features effectively. - Emphasizes important words while reducing noise from common terms. - Widely adopted in text classification tasks. | Bag-of-Words (BoW) Word Embeddings (e.g., Word2Vec) | - BoW ignores word importance and context. - Word embeddings introduce complexity and require large datasets for optimal performance. |
| **Prediction Label Decoding** | **Label Encoder** | - Simplifies encoding and decoding of categorical labels. - Ensures predictions are returned in a human-readable format. - Lightweight and easy to implement. | One-Hot Encoding | - One-Hot Encoding increases computational complexity and memory usage unnecessarily for multi-class labels. |
| **Backend Framework** | **Flask** | - Lightweight and modular, ideal for small-to-medium projects. - Easy integration with Python-based machine learning models. - Jinja2 supports dynamic web page rendering. | Django | - Django is more complex and monolithic, adding unnecessary overhead for this application. |
| **Deployment Environment** | **AWS EC2** | - Flexible configuration for compute, memory, and storage. - Elastic IP ensures consistent connectivity. - Scalable for increased traffic. - Pay-as-you-go pricing minimizes cost. | Google Cloud VM Microsoft Azure VMs | - AWS EC2 provides better integration with other AWS services (e.g., S3, VPC). - Team's familiarity with AWS services ensured ease of deployment. |
| **Network Configuration** | **Virtual Private Cloud (VPC)** | - Provides an isolated and secure environment. - Allows custom configuration of subnets, routing, and security groups. - Supports hybrid deployments. | Public Cloud Network | - Lacks the security and customization offered by VPC. - Does not allow granular control over network access and configurations. |
| **Public IP Assignment** | **Elastic IP (EIP)** | - Ensures stable public connectivity even during resource reallocation. - Essential for DNS mapping and consistent external access. - Supports dynamic association. | Dynamic IP | - Dynamic IPs change upon restart, causing inconsistency in public access and DNS configurations. |

**Conclusions**

Vulnerability detection system development and deployment showed well-planned machine learning, web application framework, and cloud infrastructure integration. The project makes use of pre-trained components such as the **Random Forest model, TF-IDF vectorizer**, and **Label Encoder** that are seamlessly integrated into a Flask application using Joblib. In textual web vulnerability prediction, this ensures accuracy, efficiency, and usability. While the Random Forest model provided a robust and understandable predictive engine, the TF-IDF vectorizer transformed textual input into meaningful numerical features. The Label Encoder improved usability by humanising numeric predictions.

Hosting the application on **AWS EC2** instances within a **Virtual Private Cloud** resulted in a secure, scalable, and reliable app deployment environment. **Elastic IP** association guaranteed sustained public accessibility, making the system available to users. These cloud installations ensured high performance, data security, and scalability while preserving cost efficiency thanks to AWS's pay-as-you-go model. Flask's interactive user interface facilitated machine learning workflow integration, decreasing Django's overhead.

Choosing ways that meet project goals and restrictions will determine success. The modular and scalable design ensures that the system may be expanded or optimised in the future by adding more complex machine learning models, improving preprocessing pipelines, or connecting new AWS services such as S3 for storage or CloudWatch for monitoring.

This project offers a robust, effective, and user-friendly web vulnerability detection solution. It stresses the importance of combining lightweight frameworks, scalable cloud infrastructure, and reliable machine learning models to create a future-proof system.

**References**

Ao, Y, Li, H, Zhu, L, Ali, S & Yang, Z 2019**,** *The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling*, Journal of Petroleum Science and Engineering, vol. 174, pp. 776–789, DOI: [10.1016/j.petrol.2018.11.067](https://doi.org/10.1016/j.petrol.2018.11.067).

Amazon Web Services (AWS) (n.d.-a) *Elastic IP addresses - Amazon Elastic Compute Cloud*. Available at: <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html> [Accessed 20 December 2024].

Amazon Web Services (AWS) (n.d.-b) *Instance types - Amazon Elastic Compute Cloud*. Available at: <https://docs.aws.amazon.com/ec2/latest/instancetypes/instance-types.html> [Accessed 20 December 2024].

Amazon Web Services (AWS) (n.d.-c) 2020**,** *Configure your VPC*, AWS Documentation, Available at: <https://docs.aws.amazon.com/vpc/latest/userguide/configure-your-vpc.html> [Accessed 20 December 2024].

Goh, H-A, Ho, C-K & Abas, FS 2022, *Front-end deep learning web apps development and deployment: a review*, Applied Intelligence, vol. 53, DOI: [10.1007/s10489-022-04278-6](https://doi.org/10.1007/s10489-022-04278-6).

Liu, C, Sheng, Y, Wei, Z & Yang, Y 2018**,** *Research of text classification based on improved TF-IDF algorithm*, IEEE Xplore , DOI: [10.1109/IRCE.2018.8492945](https://doi.org/10.1109/IRCE.2018.8492945).

Sampaio, AM & Barbosa, JG 2020**,** *Workflow scheduling with Amazon EC2 Spot Instances: building reliable compute environments*, International Journal of Machine Learning and Computing, vol. 10, no. 1, pp. 140–147, DOI: [10.18178/ijmlc.2020.10.1.911](https://doi.org/10.18178/ijmlc.2020.10.1.911).

Sangra, E, Agrawal, R, Gundalwar, PR, Sharma, K, Bangri, D & Nandi, D 2024, *Malicious website detection using random forest and Pearson correlation for effective feature selection*, International Journal of Advanced Computer Science and Applications, vol. 15, no. 8, DOI: [10.14569/ijacsa.2024.0150876](https://doi.org/10.14569/ijacsa.2024.0150876).

Singh, A & Paul, S 2020**,** *Hands-on Python deep learning for the web: integrating neural network architectures to build smart web apps with Flask, Django, and TensorFlow*, Packt Publishing Limited.

Singla, A & Chavalmane, S 2023, *Automating model deployment: from training to production*, Journal of Knowledge Learning and Science Technology, vol. 2, no. 3, pp. 340–334, DOI: [10.60087/jklst.vol2.n3.p347](https://doi.org/10.60087/jklst.vol2.n3.p347).