ABSTRACT

As technology becomes increasingly omnipresent in our daily lives, with online transactions, constant data exchanges, it is more crucial than ever to fortify our systems. With the advent of Artificial Intelligence (AI) presenting greater challenges in cybersecurity, as attackers leverage smarter algorithms to breach security measures, there is now, a pressing need for AI-driven tools to counter these threats.

This project develops a threat detection tool using Artificial Intelligence and Machine Learning techniques, designed specifically to detect Broken Access Control Vulnerabilities and Structured Query Language (SQL) Injections.

Abbreviations:

BAC – Broken Access Control

VAC – Vertical Access Control

HAC – Horizontal Access Control

MFAC – Missing Function Level Access Control

IDOR – Insecure Direct Object Reference

1. INTRODUCTION

1. 1 Background

Broken Access Control (BAC) is a critical security vulnerability that occurs when a system fails to restrict user permissions properly, allowing attackers to access highly sensitive data or functionalities. These types of vulnerabilities are commonly exploited in web applications where attackers manipulate user permissions to gain access to highly sensitive information, modify data or perform functionalities that bypasses their roles.

As per OWASP’s top ten vulnerabilities lastly released on 2021, Broken Access Control is at the top of the vulnerabilities listed, having moved from 5th position since 2017. With such an increasing surge of applicants having tested positive for this specific vulnerability, there is a need for automated tools that detects these vulnerabilities effectively. As opposed to traditional tools, a tool developed using Machine Learning techniques would be less prone to error, more adaptable, and faster.

1.2 Objective

The primary objective of the BAC Detection Tool is to develop an automated solution that can:

1. Detect different types of BAC vulnerabilities which includes Horizontal Access Control (HAC), Vertical Access Control (VAC), Missing Function-Level Access Control (MFAC), and Insecure Direct Object Reference (IDOR).

2. Detect the severity and priority level of the detected vulnerability by using relevant fields such as resource sensitivity and user role.

3. Provide detailed reports with graphical insights, including a pie chart showcasing the percentage of the composition of the vulnerabilities detected, bar chart showcasing the count of each vulnerability type detected, timeline that highlights the time of breaches and a heatmap that provides with insights on the severity and priority level.

4. Details potential causes that allows security team to find out the root cause of the error and offer mitigation strategies and recommendations to address identified vulnerabilities.

1.3 Scope

The scope of the BAC Detection Tool encompasses the following:

1. Vulnerability Detection and Classification: The tool will detect various BAC vulnerabilities, categorize them based on type, and assign severity and priority levels for each detected vulnerability.

3. Reporting and Visualization: The tool will generate comprehensive reports with graphical insights, such as bar charts, pie charts, heatmaps, and histograms, to display vulnerability data, severity distribution, and priority level distribution.

4. Export Report: The tool will be designed to support exporting reports in a CSV format for further analysis.

5. Actionable Mitigation Strategies: The tool will provide tailored mitigation recommendations for each detected vulnerability, enabling security teams to promptly address security issues.

**2. Literature Survey on Broken Access Control (BAC) Detection**

**2.1. Overview of Access Control and Vulnerabilities**

To ensure that sensitive data and functions are not accessed without permission, access control mechanisms are put in place. Over the last few decades, Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) models have emerged as effective ways of granting access permissions in an organized way, according to the user’s role or attributes.

With the rise of complex systems, Broken Access Control (BAC) vulnerabilities have become prominent, as identified by the OWASP Top 10. These include Horizontal Access Control (HAC), Vertical Access Control (VAC), Missing Function-Level Access Control (MFAC), and Insecure Direct Object References (IDOR)​(lit).

**2.2. Detection of Access Control Vulnerabilities**

* **Horizontal Access Control (HAC)**: HAC breaches occur when users access resources meant for other users. Alohaly et al. (2022) discuss using machine learning to enhance access control by categorizing user-specific permissions, which aids in preventing unauthorized horizontal access across user boundaries ([link](https://ar5iv.labs.arxiv.org/html/2207.01739)).
* **Vertical Access Control (VAC)**: VAC allows lower-privileged users to access high-level resources. Barabanov et al. (2022) demonstrate role-based checking mechanisms, suggesting how API specification processing can prevent privilege escalation ([link](https://arxiv.org/abs/2201.10833)).
* **Missing Function-Level Access Control (MFAC)**: MFAC vulnerabilities enable unauthorized users to access protected functions due to insufficient role checks. "Towards Deep Learning-Based Access Control" (2023) explores deep learning to guard sensitive functions by identifying user behavior patterns ([link](https://ar5iv.labs.arxiv.org/html/2203.15124)).
* **Insecure Direct Object References (IDOR)**: IDOR vulnerabilities arise when exposed resource identifiers allow unauthorized data access. Research emphasizes tokenization to prevent manipulation of identifiers in web applications ([link](https://ijece.iaescore.com/index.php/IJECE/article/view/33835/17135)).

**2.3. Machine Learning in BAC Detection**

* **Machine Learning Approaches**: Alohaly et al. (2022) provide a taxonomy of machine learning applications for access control, categorizing how models can improve accuracy and flexibility in detecting unauthorized access. They highlight that machine learning models, especially supervised learning, are effective for static environments, though dynamic settings still present challenges ([link](https://ar5iv.labs.arxiv.org/html/2207.01739)).
* **Random Forest for Interpretability and Efficiency**: This project’s use of a Random Forest model aligns with the need for interpretability and computational efficiency, especially in real-time systems, as opposed to the high-complexity, low-interpretability nature of deep learning models​(lit).

**2.4. Real-Time Detection and Scoring**

* **Real-Time Monitoring**: Studies like "Toward Deep Learning Based Access Control" often lack real-time capability, essential for practical applications. This project’s approach addresses this by implementing real-time detection using the Random Forest Model, which is suited for immediate vulnerability alerts in access logs ([link](https://ar5iv.labs.arxiv.org/html/2203.15124)).
* **Severity and Priority Scoring**: Few studies address vulnerability prioritization. This project’s model introduces a severity and priority scoring system to categorize threats, enhancing response efficiency. Alohaly et al. (2022) note that such an approach aligns with industry needs for prioritized security responses ([link](https://ar5iv.labs.arxiv.org/html/2207.01739)).

**2.5 Conclusion**

While existing research provides valuable insights, this project’s Random Forest-based model fills several gaps by encompassing multi-vulnerability detection, real-time capabilities, and prioritized scoring. These additions provide high-accuracy, real-world BAC detection systems.

**3. System Design**

**3.1 Overview**

The system is designed as a pipeline-based architecture tailored for detecting Broken Access Control (BAC) vulnerabilities. The architecture integrates data preprocessing, feature simulation, machine learning model training, and a real-time detection interface, displayed via a Flask-based dashboard. The primary objective is to process access logs, detect potential BAC vulnerabilities, and prioritize them based on severity and impact.

**3.2 Architecture Components and Workflow**

The system architecture consists of three main sections:

1. **Preprocessing Pipeline**
2. **Model Training and Detection**
3. **User Interface (Flask Dashboard)**

Each section is responsible for specific stages in the process, from data ingestion and preprocessing to vulnerability detection and reporting.

**3.3 Detailed Components**

**1. Preprocessing Pipeline**

* **Web Access Log Input**: The system ingests raw web access logs containing fields like client\_ip, timestamp, method, requested\_resource, status\_code, and user\_agent.
* **Preprocessing Script (**preprocessing.py**)**: Cleans and standardizes the log data, removing incomplete entries and structuring the file for compatibility with feature simulation.
* **Feature Simulation Script (**simulate\_dataset.py**)**: Adds BAC-specific fields (user\_role, resource\_sensitivity, access\_type, session\_token, user\_id, owner\_id) to simulate access control scenarios.
* **Labeled Dataset Output**: Each entry is labeled with bac\_vulnerability type (e.g., HAC, VAC, MFAC, IDOR, No\_Vuln), along with calculated severity and priority scores. This dataset is used for training the model.

**2. Model Training and Detection**

* **Model Training Script (**train\_rf\_model.py**)**: Trains a Random Forest model on the labeled dataset, optimizing it to detect BAC vulnerabilities and predict severity and priority.
* **Trained Random Forest Model**: The model is saved for deployment in the Flask app, ready to make real-time predictions on uploaded log files.

**3. User Interface (Flask Dashboard)**

* **Dashboard Interface**: Built with Flask, this interface enables users to upload files, view vulnerability detections, and analyze results.
* **Key Features**:
  + **File Upload**: Accepts CSV files for BAC vulnerability detection.
  + **Detection and Prediction**: Processes uploaded files and displays real-time predictions categorized by vulnerability type.
  + **Visualization and Reporting**: Provides insights through charts and timelines, displaying details on severity, priority, and recommended actions.
  + **Download Option**: Allows users to download detection results as a CSV for further analysis.

**3.4 Workflow**

1. **User Access and File Upload**: A security analyst uploads a CSV file via the Flask dashboard.
2. **Data Preprocessing**: preprocessing.py cleans and standardizes log entries.
3. **Feature Engineering and Simulation**: simulate\_dataset.py adds BAC-specific fields and labels entries.
4. **Vulnerability Detection**: The processed data is fed into the trained model, which classifies each entry by bac\_vulnerability type, severity\_level, and priority.
5. **Results Display and Visualization**: The dashboard displays real-time results, with graphical insights and detailed information.
6. **Report Generation**: Users can download a CSV report of detected vulnerabilities for offline analysis.

**4. Implementation**

**4.1 Data Cleaning**

To reflect real-world scenarios, a publicly available web access log dataset from Kaggle was utilized, which contained approximately 10 million entries. A representative sample of 0.01% of this dataset was selected to create a manageable, yet realistic subset. This subset was then divided into an 80-20 split for training and validation. Each log entry was parsed to extract these fields:

* client\_ip
* timestamp
* method
* requested\_resource
* http\_version
* status\_code
* response\_size
* referrer
* user\_agent
* hour
* day\_of\_week
* is\_weekend

**4.2 Data Preprocessing**

After cleaning, the dataset was enriched through a simulation script to create fields that will aid in the detection of BAC vulnerabilities. These simulated fields included:

* user\_role
* resource\_sensitivity
* access\_type
* session\_token
* user\_id
* owner\_id

These additional fields reflect typical factors involved in access control, such as user permissions, resource sensitivity, and session identifiers.

**4.3 Feature Engineering and Labeling**

A new column, bac\_vulnerability, was added to label each record according to the type of vulnerability. The five categories used for labeling, based on specific logic, were:

1. **Horizontal Access Control (HAC)**: Unauthorized access to user-specific resources.
   * **Access Type**: user\_specific, indicating the resource is tied to individual user permissions.
   * **User Role**: user or guest, meaning the user lacks elevated privileges.
   * **User ID Mismatch**: user\_id ≠ owner\_id, indicating that the resource is accessed by a user other than the owner or an authorized individual.
2. **Vertical Access Control (VAC)**: Unauthorized attempts to access sensitive resources by lower-privileged users.
   * **Resource Sensitivity**: High, signaling that access requires elevated permissions.
   * **User Role**: user or guest, indicating unauthorized access attempts by lower-privileged users.
3. **Missing Function-Level Access Control (MFAC)**: Unauthorized access to restricted operations, such as delete or modify actions.
   * **HTTP Method**: Restricted methods like DELETE, often requiring admin access.
   * **Requested Resource**: Actions such as /admin/delete\_user or /admin/modify\_data, representing administrative functionalities.
   * **User Role**: user or guest, indicating a lower-privileged user attempting restricted operations.
4. **Insecure Direct Object Reference (IDOR)**: Direct manipulation of resource identifiers, suggesting parameter manipulation.
   * **Requested Resource**: Constructed with a modified resource path, e.g., {base\_resource}/manipulated/{random\_number}, to simulate tampering.
   * **Is Manipulated**: Set to 1, indicating that the resource URL was altered.
5. **No Vulnerability (No\_Vuln)**: No specific constraints or manipulations applied, representing secure access cases.

**4.4 Severity and Priority Scoring**

To prioritize and assess vulnerabilities, two additional fields, severity and priority, were calculated:

1. **Severity Level**: Quantifies the seriousness of the detected vulnerability on a scale of 1 to 10, based on the following weighted factors:
   * **Resource Sensitivity**:
     + Low → Weight = 1
     + Medium → Weight = 2
     + High → Weight = 3
   * **Access Type**:
     + General → Weight = 1
     + User-Specific → Weight = 2
   * **Status Code**:
     + 2xx (Success) → Weight = 3
     + 4xx (Client Errors) → Weight = 1
     + 5xx (Server Errors) → Weight = 2
   * **Response Size**:
     + Small (0–1 KB) → Weight = 1
     + Medium (1–10 KB) → Weight = 2
     + Large (10 KB+) → Weight = 3
   * **BAC Vulnerability Type**:
     + HAC → Weight = 1
     + VAC → Weight = 3
     + IDOR → Weight = 2
     + MFAC → Weight = 3

**Severity Formula**:

severity\_score=(data\_sensitivity\_weight+access\_scope\_weight+status\_code\_weight+response\_size\_weight+bac\_vulnerability\_weight)severity\_score=(data\_sensitivity\_weight+access\_scope\_weight+status\_code\_weight+response\_size\_weight+bac\_vulnerability\_weight)

This score was transformed to a 1–10 range with the formula:

new\_value=2+8⋅(x−5)9new\_value=2+98⋅(*x*−5)​

**Override Condition**: If bac\_vulnerability is "No\_Vuln", severity is automatically set to 1.

1. **Priority Score**: Quantifies the priority to mitigate the detected vulnerability on a scale of 1 to 10, based on the following weighted factors:
   * **Resource Sensitivity**:
     + Low → Weight = 1
     + Medium → Weight = 2
     + High → Weight = 3
   * **Access Type**:
     + General → Weight = 1
     + User-Specific → Weight = 2
   * **Status Code**:
     + 2xx (Success) → Weight = 3
     + 4xx (Client Errors) → Weight = 1
     + 5xx (Server Errors) → Weight = 2
   * **Response Size**:
     + Small (0–1 KB) → Weight = 1
     + Medium (1–10 KB) → Weight = 2
     + Large (10 KB+) → Weight = 3
   * **BAC Vulnerability Type**:
     + HAC → Weight = 1
     + VAC → Weight = 3
     + IDOR → Weight = 2
     + MFAC → Weight = 3

**Priority Formula**:

priority\_score=(resource\_sensitivity\_weight+request\_frequency\_weight+user\_role\_weight+status\_code\_weight+response\_weight)priority\_score=(resource\_sensitivity\_weight+request\_frequency\_weight+user\_role\_weight+status\_code\_weight+response\_weight)

This score was transformed to a 1–10 range with the formula:

new\_value=y=2+0.8×(old\_value−5)

**Override Condition**: If bac\_vulnerability is "No\_Vuln", priority is automatically set to 1.

**4.5 Model Selection**

Random Forest model was selected to classify the multioutput target variables (bac\_vulnerability, priority, and severity). Random Forest was chosen for its ability to handle multiple target variables and for providing high interpretability, which is useful in analyzing access control patterns.

**4.6 Model Training**

The preprocessed and labeled dataset was used to train the model. The model was trained on features including:

* method
* status\_code
* response\_size
* user\_role
* resource\_sensitivity
* access\_type
* is\_manipulated
* is\_id\_match

Feature selection was conducted by experimenting with various combinations, retaining those that balanced accuracy with noise reduction.

**4.7 Model Evaluation**

Hyperparamter tuning was performed to maximize model accuracy. Key performance metrics, including precision, recall, and F1-score, were examined for each vulnerability type. The final model’s classification report indicated strong performance in distinguishing each category, particularly with accurate identification of HAC and VAC breaches. (include classification\_report.jpeg)

**4.8 Deployment**

The model was integrated into a Flask-based dashboard, enabling users to:

1. Upload CSV files for BAC vulnerability detection.
2. Process and analyze records through the model, displaying graphical and detailed vulnerability reports.
3. Visualize data insights with bar charts, timelines, heatmaps, and record-wise vulnerability summaries.
4. Download results as a CSV for further analysis.

**4.9 Future Work**

To extend this project, future efforts will focus on real-time processing pipelines, including automated preprocessing and log feeds to support continuous monitoring. Real-time detection and alert mechanisms would enable the system to respond proactively to ongoing BAC breaches.

**5. Results and Analysis**

**5.1 Model Performance on *bac\_vulnerability* Classification**

The model shows balanced precision, recall, and F1-score across BAC vulnerabilities.

* **Horizontal Access Control (HAC)**: Precision 0.89, recall 0.62, F1-score 0.73. Lower recall suggests some missed HAC cases, indicating a need for improved feature engineering.
* **Insecure Direct Object Reference (IDOR)**: Precision 0.88, recall 0.66, F1-score 0.75. Good detection, though recall could improve with refined features.
* **Missing Function-Level Access Control (MFAC)**: Precision and recall of 0.84, F1-score 0.86. Consistently effective in detecting unauthorized access to restricted functions.
* **No Vulnerability (No\_Vuln)**: Precision 0.88, recall 0.97, F1-score 0.92, showing strong reliability in identifying non-vulnerable cases.
* **Vertical Access Control (VAC)**: Precision 0.90, recall 0.81, F1-score 0.85, indicating effective detection of unauthorized access attempts, with a slight recall gap.

Overall accuracy is 0.87, with a macro-averaged F1-score of 0.87, reflecting balanced performance.

**5.2 Model Performance on *severity\_level* Classification**

The model achieved 0.88 accuracy in severity classification. Performance is particularly strong for high-severity cases, where accurate classification is critical for prioritizing urgent responses. This capability ensures that vulnerabilities are correctly assessed in terms of severity, providing clear guidance for addressing high-risk issues first.

**5.3 Model Performance on *priority* Classification**

The model achieved 0.94 accuracy for priority classification, with scores above 0.90 across levels, indicating excellent performance with high precision upto six digits.

**5.4 Insights from Feature Importance Analysis**

* **User Role and Resource Sensitivity**: These features ranked as the most critical for predicting BAC vulnerabilities.
* **HTTP Method and Status Code**: Moderate impact, especially for MFAC and IDOR detection. This suggests that specific request types and response codes are significant indicators of potentially malicious behavior.
* **Referrer and Timestamp**: Lower importance but provide contextual value.

**5.5 Limitations and Areas for Improvement**

* **Recall for HAC and IDOR**: The model occasionally misses HAC and IDOR cases, as shown by their lower recall scores. Feature engineering or alternative algorithms may improve detection.
* **Explainability**: Although Random Forest models provide feature importance scores, tools like SHAP values could enhance interpretability.
* **Real-Time Adaptability**: The model currently operates on static data, hence, real-time data adaptability could improve responsiveness.

**5.6 Summary of Key Findings**

The model shows high precision, effective severity, and priority classification, with opportunities to improve recall for HAC and IDOR. It is well-suited for practical BAC threat management. In key points:

* **High Precision and Low False Positives**
* **Effective Severity and Priority Classification**
* **Opportunities for Improvement in Recall**

**REFERENCES:**

1. Alexander Barabanov, Denis Dergunov, Denis Makrushin, and Aleksey Teplov. "Automatic Detection of Access Control Vulnerabilities via API Specification Processing." *arXiv*, 2022. [Link](https://arxiv.org/abs/2201.10833)
2. Mohammad Nur Nobi, Maanak Gupta, Lopamudra Praharaj, Mahmoud Abdelsalam, Ram Krishnan, and Ravi Sandhu. "Machine Learning in Access Control: A Taxonomy and Survey." *arXiv*, 2022. [Link](https://ar5iv.labs.arxiv.org/html/2207.01739)
3. Mohammad Nur Nobi, Ram Krishnan, Yufei Huang, Mehrnoos Shakarami, and Ravi Sandhu. "Toward Deep Learning Based Access Control." *arXiv*, 2023. [Link](https://ar5iv.labs.arxiv.org/html/2203.15124)
4. Ahmed Anas, Salwa Elgamal, and Basheer Youssef. “Survey on detecting and preventing web application Broken Access Control Attacks.” *International Journal of Electrical and Computer Engineering (IJECE)*, 2024. [Link](https://ijece.iaescore.com/index.php/IJECE/article/view/33835/17135)
5. Nabila Farnaaz. "Random Forest Modeling for Network Intrusion Detection System." *Procedia Computer Science*, ScienceDirect, 2016. [Link](https://www.sciencedirect.com/science/article/pii/S1877050916311127)