**Problem code- J-001 - Programming Language -Java/Python**

**Cyber Security - Problem Statement**

Get me vulnerable security issues in my application and recommended fixes for a Maven build application.

**Brief:**  
Security vulnerabilities in applications can lead to severe consequences, including data breaches, unauthorized access, and financial loss. Identifying and fixing these vulnerabilities early in the development lifecycle is crucial. This solution aims to scan Maven-based applications for security issues and provide recommended fixes to ensure robust application security.

Please use the following portal to download the application:

* App Exchange Portal: <https://apps.xforce.ibmcloud.com/>
* Direct App URL for download: <https://apps.xforce.ibmcloud.com/extension/6f5cc6de1e5e2dad38bfa755c3f2b80b>

Students will need to create their own IBMid to download the app ZIP file, you can register with any email id on the portal(that’s called IBMid).

**Outcome:**

* **Create similar tool as mentioned in the example tools**
* Identify security vulnerabilities in a Maven-based application.
* Generate a detailed report highlighting the detected vulnerabilities.
* Provide recommendations or automated fixes for each detected issue.
* Ensure improved application security with minimal developer effort.
* Seamless integration with CI/CD pipelines for continuous security assessment.

**Example Tools:**

* Mend
* OWASP Dependency-Check
* Snyk
* SonarQube
* Trivy
* GitHub Dependabot
* Maven Enforcer Plugin

**Constraints:**

* The solution must work within a Maven-based application.
* It should support offline and online security scanning modes.
* Should not introduce excessive build-time overhead.
* Must provide clear and actionable remediation steps.
* Should be compatible with Java applications built on JDK 8 and above.

**Key Features:**

* Automated scanning for security vulnerabilities in dependencies.
* Integration with existing Maven build workflows.
* Generation of human-readable reports with risk classifications (Critical, High, Medium, Low).
* Support for both static analysis and runtime vulnerability detection.
* Ability to apply security patches automatically where possible.
* Configurable scanning rules based on enterprise security policies.
* Integration with DevSecOps tools for continuous security monitoring.

This document serves as the foundation for building a security vulnerability scanner and remediation tool tailored for Maven-based applications. Let me know if you need refinements!

**Problem code- JP-001- Programming Language - Java/Python -**

**Problem Statement** - Design and develop a Java/Python application to identify and remove duplicate applications based on content (not based on Filename, file type, or timestamp). Also organize the applications into categories based on pre-defined rules.

**Brief:** The aim of this project is to build an intelligent application management tool that can identify and eliminate redundant application installations by analyzing their content rather than relying on superficial file system attributes. This application will employ content-based comparison techniques to accurately detect duplicates, even if they have different filenames, types, or timestamps. Furthermore, it will incorporate a rule-based categorization system to organize the identified applications into predefined categories, providing a structured and manageable overview of the installed software.

**Outcome:** **A Java or Python application capable of identifying and removing duplicate applications based on content analysis and organizing applications into categories according to pre-defined rules**.

**Tools:**

* **Java:**
  + Java Development Kit (JDK)
  + Apache Commons IO (for file handling)
  + Hashing libraries (e.g., Apache Commons Codec for SHA-256/MD5)
  + Rule engine (e.g., Drools - optional for more complex rules)
* **Python:**
  + Python 3.x
  + os module (for file system interaction)
  + hashlib (for generating content hashes)
  + json or yaml (for defining categorization rules)
  + Rule engine (e.g., rule-engine - optional for more complex rules)

**Constraints:**

* Duplicate identification must be based on the content of the application files (e.g., executable code, libraries) and not solely on metadata.
* The application should provide a mechanism to define and manage pre-defined categorization rules.
* Users should be presented with a list of identified duplicate applications and have the option to select which ones to remove.
* The categorization process should be automated based on the defined rules.
* The application should handle different application file structures and sizes efficiently.

**Key Features:**

* **Content-Based Duplicate Detection:** Generates cryptographic hashes (e.g., SHA-256) of application files to identify duplicates regardless of filename, type, or timestamp.
* **Rule-Based Categorization:** Allows users to define rules based on application metadata (e.g., extracted information from manifests, file paths, or potentially basic content analysis) to automatically categorize applications.
* **Duplicate Removal:** Provides a user interface or command-line option to remove selected duplicate applications.
* **Categorized Application Listing:** Presents a clear and organized view of the installed applications, grouped according to the defined categories.
* **Configuration Management:** Enables users to configure the directories to scan for applications and manage the categorization rules.
* **Logging and Reporting:** Provides logs of the duplicate identification, removal, and categorization processes.

**Problem code- C-001- Programming Language - C++ Problem Statement**

**Airline Reservation System with Dynamic Pricing**

In this assignment, you are required to develop a console-based Airline Reservation System. The system must support booking, modifying, and cancelling flight tickets for various flights. The core focus is on implementing dynamic pricing, where the price of tickets varies based on the number of seats available and how early/late the booking is made. This simulation will model real-world airline ticket pricing strategies and also ensure thread-safe access for multiple users interacting with the system concurrently.

**Key Functionalities to Implement:**

1. Booking a Ticket:
   * User provides origin, destination, date, and time.
   * System checks seat availability in real-time.
   * Price is calculated dynamically based on current demand and timing.
   * Generates a unique booking ID and confirms the ticket.
2. Modifying a Ticket:
   * User can change the flight, time, or passenger details.
   * Updated price is shown and recalculated.
   * System adjusts seat availability accordingly.
3. Cancelling a Ticket:
   * User can cancel a ticket using the booking ID.
   * Refund logic can be basic or time-based.
   * Cancelled seat becomes available again.
4. Dynamic Pricing Logic:
   * Ticket price increases as the number of available seats decreases.
   * Closer booking dates to departure increase the ticket price.
   * Base price + surge based on (1 - available seats / total seats) and days to departure.

**Expected Outcomes:**

1. A fully functional ticket booking system
2. Real-time seat tracking
3. Dynamic pricing implementation reflecting real airline behaviour
4. Demonstrated thread-safe code to support multi-user simulation

**Tools & Technologies:**

* Programming Language: C++
* Data Structures:
  + Priority Queue for managing booking priority (e.g., for upgrades, cancellations, etc.).
  + File Handling for storing ticket and user data persistently.
* Concurrency:
  + Use of C++ threading and mutexes to ensure thread safety.
  + Simulate multiple users booking tickets simultaneously.

**Constraints & Requirements:**

1. Real-Time Availability Checking:  
   Your system must always show accurate seat availability.
2. Dynamic Pricing Model:  
   Incorporate time-of-booking and demand (seats left) to calculate price. Consider:
   * Base Fare + Surge Price (depending on demand).
   * Fare increases as the number of days before departure decreases.
3. Thread Safety:
   * Use mutex locks and synchronization where necessary.
   * Demonstrate with a simulation where multiple threads attempt to book tickets at the same time.
4. No Use of STL (optional unless specified otherwise):
   * Implement your own priority queue or manage with basic arrays/pointers.

**Advanced Suggestions (Optional but Encouraged):**

1. Simulate frequent flyer points or user tiers (Gold/Silver).
2. Include meal selection, baggage add-ons at booking time.
3. Store booking data in files and load into memory at startup.
4. Generate a report of most popular routes or busiest flights.

**Check your solution for :**

1. Code Design and Modularity
2. Correctness of Dynamic Pricing Logic
3. Proper Use of OOP Concepts (Classes, Inheritance, Encapsulation)
4. Efficient Memory Management and Thread Safety
5. Clarity of Menu and User Interaction
6. Creativity in Features Beyond Requirements

**Output:**

1. Display the menu for the user to choose an option

* Also include the sample images while code submission (User providing inputs and getting the desired output)

**Problem code- P-002 - Programming Language – Python with AI/ML-**

**Intelligent office access management system**

**Brief:**

The application shall use AI-powered object detection to accurately identify the employee upon swiping for entry into the office. If the employee forgets their access card, the office camera should reliably recognize them using face detection and body dimensions, ensuring secure access to the office premises.

**Outcome:**

A database will be created to store comparable employee details using face detection and body dimensions.

**Tools:**

* Best possible, least expensive and high security software.

**Constraints:**

* Data should be collected and stored in elastic database which scales up automatically.
* Data stored must be highly secure and confidential
* Data captured between the multiple cameras should be interlinked or joinable.
* Malfunctioning cameras should be automatically reported for repairs.

**Key Features:**

**Employee identification:**

Employees should be correctly identified and allowed appropriate access.

**Identifying tailgating:**

Identify if the correct person is using the access card and detect any tailgating.

**Problem code- P-003- Programming Language - Python/Flask/API –**

**Problem Statement**

Design and **develop a pluggable caching abstraction layer** (preferably API) to use any of the caching framework like Redis, Memcached etc - Allow switching between Redis, Memcached, or other caching systems without changing the API.

**Brief:** The aim of this project is to build a flexible and adaptable caching solution for applications. This layer will abstract the underlying caching framework, enabling seamless integration with various technologies such as Redis, Memcached, and others. The core principle is to provide a consistent API that developers can use without needing to be concerned about the specific caching implementation. This abstraction will facilitate easy switching between caching systems for reasons such as performance optimization, cost reduction, or leveraging specific features of a particular framework, all without requiring significant code changes in the application layer.

**Outcome:** A robust and pluggable caching abstraction layer (API) that allows applications to interact with different caching frameworks (e.g., Redis, Memcached) through a unified interface, enabling easy switching between these frameworks without modifying the application's core caching logic.

**Tools:**

* Python (preferred for API development and flexibility)
* Flask or FastAPI (for building the API)
* python-redis (for Redis integration)
* pymemcache (for Memcached integration)
* Abstract Base Classes (ABCs) or Interfaces (for defining the caching API contract)
* Dependency Injection patterns (for managing caching framework implementations)

**Constraints:**

* Develop a clear and consistent API for common caching operations (e.g., set, get, delete, clear).
* Implement concrete implementations for at least Redis and Memcached.
* Design the architecture to easily accommodate the addition of new caching framework integrations in the future.
* Ensure thread safety and handle potential connection errors gracefully.
* The switching of the underlying caching framework should ideally be configurable (e.g., via environment variables or configuration files) without requiring code recompilation.

**Key Features:**

* **Unified Caching API:** Provides a consistent set of methods for interacting with the cache, regardless of the underlying implementation.
* **Pluggable Framework Support:** Allows for easy integration of different caching frameworks by implementing the defined API contract.
* **Runtime Switching:** Enables the application to switch between configured caching frameworks (e.g., Redis to Memcached) through configuration changes.
* **Abstraction of Framework Specifics:** Hides the unique details and complexities of each caching framework from the application code.
* **Extensibility:** The design should be easily extendable to support additional caching functionalities or framework-specific features in a non-breaking manner.

### **Problem code- P-004 - Programming Language -Python with AI/ML** –

### **Problem Statement**

### **Complaint Management System with Chatbot Integration & Ticket Support Generation**

**Brief:**

This project aims to develop an intelligent complaint management system integrated with a smart AI-based ticketing system (Severity Level). The chatbot will try to resolve user complaints & if not able to resolve it will log complaints via the chatbot interface, and each complaint will generate a ticket. The AI will intelligently categorize, prioritize, and assign tickets based on the nature and urgency of the complaint. The system will feature real-time tracking for users and an admin dashboard for monitoring, assigning, and resolving complaints efficiently. The goal is to streamline the entire process using automation, artificial intelligence, and a responsive chatbot interface.

**Outcome:**

A sophisticated complaint management system where users can log and track complaints through an AI-powered chatbot, with an intelligent ticketing system that prioritizes and assigns tasks for faster resolution.

**Tools:**

* Any platform
* AI/ML Libraries (e.g., TensorFlow, Scikit-learn)
* Natural Language Processing (NLP) Libraries (e.g., NLTK, SpaCy, Dialogflow/Rasa)
* Chatbot Framework (e.g., Dialogflow, Rasa)

**Constraints:**

* AI-powered ticket generation, categorization, and prioritization
* NLP for chatbot-based user interaction
* Real-time complaint status tracking
* Admin dashboard for complaint and ticket management
* Intelligent assignment of tickets based on complexity, urgency, and team availability
* Notifications and automated responses

### **Key Features:**

1. **User Registration & Authentication:**
   * Secure user authentication to create profiles and log in, ensuring data privacy and personalized complaint management.
2. **AI-Powered Chatbot Integration:**
   * A chatbot will act as the first point of contact for users, helping them log complaints, answer FAQs, and guide them through the system.
   * NLP models will enable the chatbot to understand user inputs in natural language, making the complaint registration process seamless.
   * The chatbot can provide users with real-time updates on the status of their complaints.
3. **AI-based Ticketing System:**
   * **Automatic Ticket Creation:** Upon registering a complaint, the system automatically generates a ticket with all relevant details.
   * **Intelligent Categorization:** AI will categorize tickets based on predefined categories (e.g., billing, technical, service) using machine learning models.
   * **Prioritization System:** AI will assign priority levels (urgent, high, medium, low) based on the complaint's nature and keywords (e.g., "urgent" or "service outage").
   * **Dynamic Assignment:** The system will use AI to intelligently assign tickets to the relevant teams or agents based on expertise, workload, and availability.
4. **Complaint Tracking & Status Updates:**
   * Users will receive real-time updates on the progress of their complaints, either through the chatbot interface or via email/SMS notifications.
   * The system will allow users to view complaint status such as "Registered," "In Progress," "Under Review," or "Resolved."
5. **Admin Dashboard with Analytics:**
   * **Ticket Overview:** Admins can view a dashboard with all open and resolved tickets, prioritized by urgency.
   * **AI-Assisted Assignment:** The system will suggest which team or agent to assign based on skill sets and current workloads.
   * **Detailed Analytics:** Generate reports on response time, resolution time, number of complaints handled, user satisfaction, etc. AI can help identify patterns (e.g., recurring issues) and suggest improvements.
6. **Notifications & Escalation Mechanism:**
   * **Automated Notifications:** Users will receive updates on the status of their tickets at key stages (e.g., ticket creation, assignment, progress, resolution).
   * **Escalation Workflow:** For unresolved or high-priority issues, the system will automatically escalate the ticket to higher-level management or specialists after a certain time threshold.

**Problem code- P-005- Programming Language – Python with AI/ML**

**Job Portal with AI-based Resume Screening & Career Counseling**

**Brief:**

This project focuses on developing a job portal with AI-powered resume screening capabilities and personalized career recommendations to students by analyzing their skills, interests, and aptitudes. Users can create profiles, upload resumes, search for jobs, and receive feedback on their resumes through an AI-driven system. Employers can post job openings and screen applicants using AI algorithms to match candidate resumes with job requirements. Additionally, users will receive career counselling assessing their strengths, skills, interests, and provide support, and resources to navigate career transitions and development. This AI-powered solution should offer tailored career suggestions and actionable insights to help students align their strengths with suitable career paths based on inputs from resume.

**Outcome:**

An intelligent job portal that streamlines the job search process with AI-enhanced resume screening and personalized career counselling.

**Tools:**

* Any platform
* AI/ML Libraries for resume analysis and matching algorithms

**Constraints:**

* AI-powered resume analysis and feedback
* User authentication and secure profiles
* AI-based job matching and screening for employers
* Customized career counselling for users based on resume analysis
* Resume parsing, skill gap analysis, and tailored career advice

**Key Features:**

**AI-driven Resume Screening:**

The system uses AI models to analyze resumes and provide feedback on formatting, skill gaps, and suggestions for improvement.

**Job Search & Application:**

Users can search for jobs, apply online, and track their application status.

**Employer Portal for Job Posting:**

Employers can post job listings, review candidate profiles, and utilize AI to match job requirements with suitable applicants.

**AI-based career counselling:**

The AI system will help understand the best available opportunities and career path based on how well their skills match the job description, improving the hiring process.

**Problem code- P-006 - Programming Language – Java/Python**

**Cyber security Problem Statement**  
**Predictive Risk Scoring Assessment**

:pushpin:**Brief**  
Insider threats are among the most critical challenges for any organisation. Identifying them at an early stage helps prevent data breaches, minimise downtime, and avoid financial loss.

:dart:**Outcome**

* Assign dynamic risk scores to specific entities
* Identify and surface rules and patterns that most significantly impact each entity
* Dynamically determine risk scores in the range of 5 to 50 and assign them automatically
* Provide actionable recommendations, such as adjusting configurations or tuning firewall rules
* Improve the overall accuracy of detecting true threats and reduce false positives through better scoring and recommendations

:hammer_and_wrench:**Tools & Technologies**

* **Python**
* **Machine Learning Models**
  + Isolation Forest
  + Random Selection
* **React JS**

:sparkles:**Key Features**

* Dynamic, data-driven risk scoring that adapts over time
* Better visibility into user and entity health, highlighting which rules and patterns contribute most to risk
* Granular-level assessment and fine-tuned risk scoring for each rule, helping to identify early threats so users can take timely action

**Problem code- C-002 - Programming Language – C++**

**Problem Statement - A Custom Memory Allocator using C/C++**

**Brief:**

Standard dynamic memory allocation functions like ***malloc, free, new,*** and ***delete*** are provided by system libraries and offer limited control over memory management strategies. These generic allocators may sometimes lead to performance issues such as fragmentation, poor cache utilization, or lack of thread safety in specialized applications.

This project aims to implement a custom memory allocator in C/C++ that replicates the core functionality of ***malloc*** and ***free***. The allocator will manage a fixed-size memory pool (statically allocated or heap allocated), handle allocation and deallocation requests, and optimize memory reuse. It will explore techniques such as free lists, block splitting/merging, and allocation strategies (e.g., first fit, best fit, or any other strategy you think of) to improve efficiency and reduce fragmentation.

**Outcome:**

An efficient and modular custom memory allocation system written in C/C++ which minimizes fragmentation and maximizes the re-use of freed memory.

**Tools:**

* Linux OS
* C/C++ compiler
* Makefile

**Constraints:**

* Only C/C++ can used as the programming language
* Use a pool of memory of 2MB (2 \* 1024 \* 1024 bytes). You can allocate it as a byte array, either on stack or on heap, and use it as your free memory pool.

#define MAX\_POOL\_SIZE 2 \* 1024 \* 1024; // 2 MB size  
char mem\_pool[MAX\_POOL\_SIZE];  
OR  
char\* mem\_pool = (char\*)(malloc(MAX\_POOL\_SIZE));

* Each memory block should be of 1KB in size (1024 bytes). So, a total of only 2048 blocks of memory can be accommodated in the pool.
* There is no constraint of not using the C++ STL library but for this project they won’t be of much use as the memory management has to be designed by you. By using STL containers like vector<> or list<>, you will not have control over the internal memory allocation and deallocation. (e.g., vector allocates twice the amount of memory you ask for, so it might use up all the free memory pool space you have). Moreover, STL containers will not allocate memory from your custom pool of 2MB space.

**Key Features:**

**Custom Pool of Memory:**

Provide a low-level control over memory allocation and deallocation by managing a statically allocated (OR) dynamically allocated pool of memory.

**Allocation and De-allocation Strategies:**

The custom memory allocator should track the number of allocated and free blocks of memory and use some allocation strategies like ***first-fit***(first block that’s big enough)***, best-fit*** (smallest suitable block)***, buddy-system*** (power-of-two splitting), or any other allocation strategies one can think of.

**Fragmentation and De-fragmentation:**

Effectively manage fragmentation and de-fragmentation of memory blocks, i.e., splitting the large free memory pool into blocks while allocation and merging them while de-allocation.

**Memory Leak Checker**

As the system will keep track of allocations and de-allocations of the memory blocks, implement a feature of memory leak checker, which when asked for, will dump the memory leak information with memory blocks metadata (number of blocks, block index, etc.).

**Testing**

* Allocating some random sized memory blocks and then freeing some of them.
* Checking if the freed memory blocks are being re-used
* Test with some fragmentation scenarios
* Run memory leak checker

**Few Hints:**

* Maintaining a linked-list of memory blocks (1K size each).
* Each memory block of 1K can be a node in the linked list, with some bytes in the node used for block-header to maintain metadata of the block.
* Implementing functions like ***xmalloc(), xfree(),*** which when invoked by user would take some memory from the free pool instead of calling malloc(), free().

Design some class hierarchies for better handling of objects.