MoRocco – A Secure Exam Operating System for Academic Integrity

Rocco Guevara, Dr. Aravind Mohan

McMurry University, Abilene, Texas

Abstract

Academic institutions increasingly face challenges in maintaining exam integrity as students gain access to advanced AI tools, remote assistance technologies, and operating-system–level workarounds. Traditional lockdown browsers restrict applications at the software layer but remain vulnerable because they depend on the security of the host environment. As a result, universities require a more robust solution that isolates the testing environment from the underlying system while still supporting programming tasks, technical assessments, and classroom-scale deployment. In this project, we introduce MoRocco, the Modular Restricted Operating Computing Core OS, a lightweight secure exam operating system designed specifically for controlled academic testing. To support modern computing courses, MoRocco incorporates an Object-Oriented Programming approach that improve maintainability and extensibility. Additional supporting utilities are developed using Python and shell scripting to enable efficient integration with Linux kernel features and provide proper restrictions. The operating system provides only essential tools such as a code editor, compiler, and terminal, all governed by a strict security policy that validates binaries and enforces a zero-trust runtime. MoRocco can run locally on low-spec university laptops as a virtualized or containerized image, eliminating dependency on network speed or cloud resources. Instructors prepare exam materials through a secure packaging utility, ensuring that exam files, datasets, and auto-grading scripts become available only after the OS boots and remain isolated from the host. Preliminary evaluations demonstrate that MoRocco offers a tamper-resistant, resource-efficient, and pedagogically flexible testing environment that surpasses traditional lockdown solutions.

Introduction

Academic institutions face growing challenges in maintaining exam integrity due to the increasing availability of advanced AI tools, remote assistance technologies, and operating system–level workarounds. Traditional lockdown browsers restrict access at the software layer but remain vulnerable because they rely on the security of the host environment. According to research by EDUCAUSE Review, students can often bypass conventional proctoring mechanisms using system-level exploits or external devices, raising concerns about fairness and credibility in assessments. Similarly, a study by the International Organization for Standardization highlights that secure exam environments must consider both technological and human factors to prevent academic misconduct effectively. In many computing courses, instructors require students to compile code, run programs, and access restricted tools during exams, which traditional lockdown solutions cannot support safely. The challenge lies in designing a system that provides robust isolation, enforces strict policies, and still allows essential exam functionality. To address these issues, we introduce MoRocco: Modular Restricted Operating Computing Core OS, a lightweight secure exam operating system designed to safeguard academic integrity. MoRocco integrates a modular Java framework for core components such as file and process isolation, policy enforcement, and exam launcher management, while supporting Python and shell scripts for auxiliary utilities. The system boots into a sandboxed Linux environment, disabling internet access, preventing unauthorized application execution, and restricting host file access, providing a secure toolset, including a code editor, compiler, and terminal, so students can complete programming exams without compromising security. Additionally, MoRocco incorporates audit logging, network controls, and an auto-grading engine to streamline instructor workflows and enhance accountability, while also promoting maintainability through modular architecture and layered policy enforcement. Through this research, we aim to answer questions such as how a secure OS can balance isolation with usability for programming exams, what system-level mechanisms are most effective for preventing tampering, and how modular design improves extensibility of secure exam environments. Our experiments demonstrate that MoRocco provides a tamper-resistant, resource-efficient, and pedagogically flexible platform addressing the limitations of traditional lockdown solutions while supporting a full range of exam requirements, with consistent performance across varied hardware configurations and minimal disruption to standard programming workflows, suggesting that a purpose-built operating system approach can strengthen trust in digital assessments while preserving practicality for computing education.

Methodology

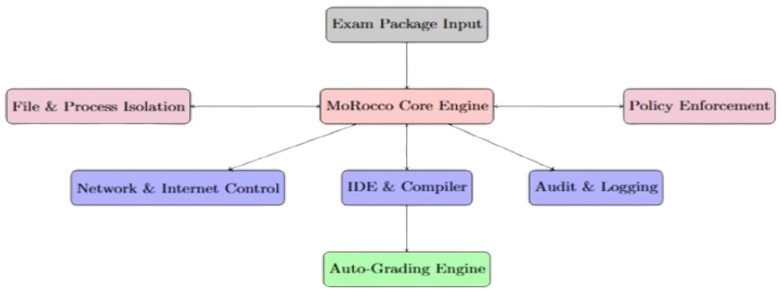


Figure 1 Architecture of MoRocco

The MoRocco system integrates core operating system modules, security controls, and exam utilities to create a tamper-resistant testing environment. It uses a layered architecture to manage exam input, enforce policies, isolate files and processes, provide secure tools, control network access, and support automatic grading, as shown in Figure 1. The MoRocco methodology organizes the assessment process around secure, automated evaluation. The tool is built using Java object-oriented programming, Linux bash scripts for automation, and QEMU to run an Alpine Linux virtual machine. Table 1 shows a sample exam package developed for a Java course. Student submissions are executed in isolated environments to prevent interference or tampering. The algorithm in Table 2 explains how exam packages allow students to complete exams safely and efficiently. After answering the questions, students run the autograding module included in the exam package. This process generates the completed exam package, autograder results, and execution logs, as shown in Table 3.

Table 1: Sample Exam Package

Question 1: Palindrome Check Write a Java method isPalindrome(int n) that returns true if the given integer is a palindrome and false otherwise. Enter 121 to test.	**Question 2: Object-Oriented** Design a Java class named Student with the following **private fields**: String name int id double gpa	**Question 3: Second Largest** Write a Java method secondLargest(int[] arr) that returns the second largest integer in the array. Use an array with 4,9,1,7 to test
Result: true	Result: Student{name='Bob', id=5, gpa=3.8}	Result: 9

Table 2: MoRocco Algorithm

Table 3: MoRocco Output

Input: Exam Package P Output: Complete Exam Package C', Execution Logs L 1) M ← Setup Alpine Linux Machine using QEMU 2) Initialize isolated network configuration for M (no external access) 3) Transfer P into M from Host 4) Verify integrity of P using checksum validation 5) P' ← Decrypt(P) using OpenSSL 6) Validate digital signature of P' (if applicable) 7) Student ← Workspace(P') 8) Set restricted permissions for Student environment 9) Initialize system monitoring tools inside M 10) L ← ExecutionLogs.Start(Student) 11) Capture system state snapshot before execution 12) C ← Complete Exam (P', Student) 13) Continuously monitor CPU, memory, and file access during execution 14) Record all file system changes made by Student 15) Detect and log any policy violations or abnormal behavior 16) L ← ExecutionLogs.Stop(Student) 17) Generate integrity hash of C 18) C' ← Encrypt C using OpenSSL 19) Transfer C' and L from M to Host 20) Securely destroy temporary files and keys inside M 21) Shut down and terminate M 22) return Host(C', L)	**Autograder** Question 1: PASS Question 2: PASS Question 3: FAIL **Execution Logs** cd workspace ls vi Palindrome.java javac Palindrome.java java Palindrome vi Student.java vi Stub.java javac Student.java javac Stub.java java Stub vi Largest.java javac Largest.java java Largest cd grader ls ./autograder.sh poweroff

Experiments

Beyond security enforcement, MoRocco was evaluated for execution correctness, grading reliability, and performance overhead during live programming exams. As shown in Table 4: Distribution of Programming across Images, student submissions were consistently executed across multiple pre-configured exam images, demonstrating stable and uniform behavior without configuration-related discrepancies. Table 5: Security Enforcement Outcomes During Exams confirms that policy violations were effectively blocked while legitimate compilation and execution proceeded without false positives or grading interruptions. Furthermore, Figure 2: Performance Overhead Compared to Native Execution illustrates that the additional isolation and monitoring mechanisms introduce only minimal latency relative to a native system, confirming that MoRocco preserves operational efficiency while maintaining exam integrity.

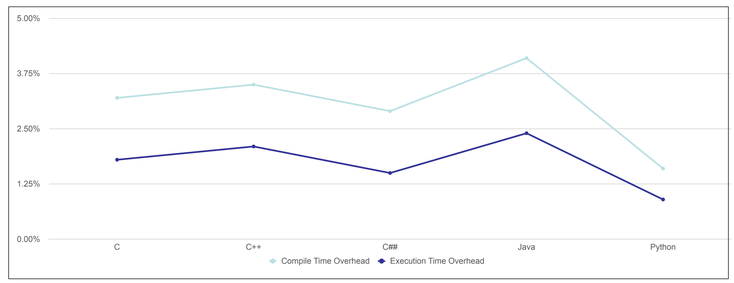
Table 4: Distribution of Programming across Images

Programming Course	Number of Exams	Primary Focus Area
C	5	Pointers, compilers, scope, execution order
C++	5	References, object lifetime, compilation/runtime
C#	5	Value vs references typos, exceptions, scope
Java	5	Object references, compilation errors, execution flow
Python	5	Name binding, mutability, runtime errors
Total	**25**	

Table 5: Security Enforcement Outcomes During Exams

Security Event	Detected	Blocked
Unauthorized application launch	Yes	Yes
Host file system access atempt	Yes	Yes
Clipboard usage attempt	Yes	Yes
Netwowkr access request	Yes	Yes
Binary integrity violation	Yes	Yes

Figure 2: Performance Overhead Compared to Native Execution



Conclusions and Future Work

These experiments demonstrate that MoRocco is not only secure, but also correct, deterministic, and efficient. The system preserves language semantics, ensures reproducible grading, introduces minimal performance overhead, and scales to realistic academic workloads. Together with the security evaluation, these results establish MoRocco as a robust and reliable operating system for high-stakes programming examinations.

Contact Us

Rocco Guevara email: guevara.rocco@mcm.edu

Dr. Aravind Mohan email: mohan.aravind@mcm.edu
Phone: (325) 793-3845