Integrity Models Lab Report

Gabriel Tinsley

CS 331 - 002

February 22, 2025

Lab 1: Implementing the Biba Integrity Model

1. **Objective** – What the lab aimed to achieve.

The objective of this lab was to implement the Biba Integrity Model and output the test cases demonstrating "No Read Down" enforcement and "No Write Up" enforcement.

2. **Implementation Steps** – A brief **step-by-step explanation** of how they implemented the integrity model.

Given the example python code, I first split the combination of user and file integrity into just user integrity and defined the ACM as the usual user and file and rights. Then I mapped the integrity levels to values for "Low" that is 1 and "High" that is 2. The next thing I did was get the file integrity then I finally checked the access.

3. **Code Explanation** – Summarize **key functions and logic** in their Python/Linux implementation.

Key functions in my code are:

(1) Get_file_integrity: This function allows me to retrieve the integrity level of a file from any user entry. This finds the first occurrence of the file and defaults to the lowest level if the file isn't explicitly listed.
(2) Check_access: This function gets the users integrity level and the files integrity level. Through an if-else statement I am able to incorporate the Biba integrity model of No Read Down and No Write Up rules.

4. **Results & Observations** – What they learned from their tests.

My tests taught me how to check the integrity levels of my subjects and objects. I failed many of my tests before finally accomplishing the goal of No Read Down and No Write Up. I put my test results from the command line into biba_test_results.txt document. The tests with Linux taught me how to attempt to modify files from the command line and I copied the results into a security_log.txt document.

5. **Challenges & Solutions** – Any difficulties encountered and how they solved them.

Difficulties I encountered were passing the tests. I ran the python debugger many times and that helped me to realize I needed to change how I was using the > or < operator. With keeping track of my integrity levels as integers I was able to successfully pass all my tests once I solved my problem.

Lab 2: Simulating the Low-Water-Mark Policy

Step 4: Observe integrity degradation

- **Users progressively lose access** to high-integrity data. **Discuss why this model may be impractical in real-world systems**.

In the Low Water Mark policy users lose access to high-integrity data which is impractical in real-world systems because there is no way for users to gain integrity once lost. The fact that a user permanently loses access to data once dropped is very inefficient. In a real-world system people would avoid reading certain files to not lose their integrity.

==========

1. **Objective** – What the lab aimed to achieve.

Lab 2 aimed to achieve simulating the Low-Water-Mark Policy, which updates the user integrity level to the minimum when reading.

2. **Implementation Steps** – A brief **step-by-step explanation** of how they implemented the integrity model.

I started by creating a simple ACM with subjects Alice with Integrity "High" and Bob with Integrity "Low" and object File1 with integrity "Low". I then mapped the integrity levels to 1 meaning "Low" and 2 meaning "High". The function low_water_mark_read takes a user and file then updates the user's integrity to a lower level of the file. The function low_water_mark_write takes a user and a file and checks if the user has write access. Access is denied if the user's integrity level is lower than the file's.

3. **Code Explanation** – Summarize **key functions and logic** in their Python/Linux implementation.

Key functions in my code are:

(1) Low_water_mark_read: This function updates user integrity level to the minimum when reading
(2) Low_water_mark_write: Denies write access if the user's integrity level is lower than the file's

4. **Results & Observations** – What they learned from their tests.

I tested my functions to ensure they are working properly. I noticed that because File1's integrity is "low", when Alice with "high" integrity reads the file her integrity falls to "low". And when working from the command line, after dropping Alice's integrity, when she attempts to modify File2 with integrity "high" that gets denied because Alice's integrity level is now "low". Finally, I made my own test to make sure my low_water_mark_write function actually allows users with high integrity to modify files and it works correctly.

5. **Challenges & Solutions** – Any difficulties encountered and how they solved them.

One challenge I had after I changed Alice's integrity to low by mapping 1 and 2 for low and high was having Alice's integrity go back to a string value from a number. Eventually I found the solution as to have an if statement that checks if Alice's integrity is equal to 1 and change that back to a string if true.

Lab 3: Implementing the Ring Policy

1. **Objective** – What the lab aimed to achieve.

Enforce the Ring Policy because subjects can read all objects, but subjects cannot modify higher-integrity data.

2. **Implementation Steps** – A brief **step-by-step explanation** of how they implemented the integrity model.

I defined my ACM and mapped the integrity levels to values 1 for low, 2 for medium, and 3 for high. My function ring_policy takes a user, file, and action and gets the user_level and file_level as integers. Then an if-else statement compares user_level and file_level to grant or deny access.

3. **Code Explanation** – Summarize **key functions and logic** in their Python/Linux implementation.

The key functions are:

(1) Ring_policy: This function implements the ring policy, all can read but only high-integrity can modify.

4. **Results & Observations** – What they learned from their tests.

From running the tests I found my ring_policy function to be working correctly. All the users can read the files but only high-integrity users can modify them. All the tests passed correctly. I had trouble figuring out the test execution control in linux.

5. **Challenges & Solutions** – Any difficulties encountered and how they solved them.

A challenge I had was comparing the user level and file level. When I turned the user level and file level into integers, I found this to be the solution to the comparison issue.

Lab 4: Real-World Case Study

1. **Objective** – What the lab aimed to achieve.

An analysis of a real-world security incident (SolarWinds attack) involving integrity compromise. And bash script implementing an integrity enforcement technique.

2. **Implementation Steps** – A brief **step-by-step explanation** of how they implemented the integrity model.

Through online research I found an article about the SolarWinds attack. I was very confused with creating a bash script, I have never worked with a bash script before and finding how to implement the integrity enforcement technique is beyond my knowledge. Although, looking at the example given I see that bob is an untrusted user so it would make sense for him to be denied access.

3. **Code Explanation** – Summarize **key functions and logic** in their Python/Linux implementation.

My code was straight from the example given, I have no key functions or logic to explain.

4. **Results & Observations** – What they learned from their tests.

From my research I learned how important it is to detect unauthorized users and users granting themselves access to things. The SolarWinds attack is a big eye opener that was not caught for months from just a simple software update that had malicious code in it.

5. **Challenges & Solutions** – Any difficulties encountered and how they solved them.

I felt a big challenge with writing in shell script, I have never used shell script before and the instructions on what to do with the shell script are hard to follow.

Reflection Questions

1. **Why do organizations need different integrity models?**

Organizations need different integrity models because security needs are based on regulations, industry, and risks. Biba or Bell-LaPadula models are approaches that enforce data integrity and prevent unauthorized modifications.

2. **Which integrity model is best suited for modern cloud environments? Why?**

The Biba model is best suited for modern cloud environments because it enforces “No read up” and “No write down” ensuring that only authorized users can modify objects. This helps prevent data corruption and unauthorized access in cloud environments.

3. **What is the biggest challenge when enforcing integrity policies in an operating system?**

One of the biggest challenges is balancing security and usability. Strict integrity policies like Low-Water-Mark Policy can restrict user operations leading to no way for users to gain integrity after losing it.

4. **If you were designing an access control system, which integrity model would you use and why?**

I would use the Biba model because it ensures user and object integrity when modifications occur. It follows closely with role-based access control (RBAC) and modern security best practices.

5. **How could an attacker try to bypass integrity models, and how would you prevent it?**

Hackers could try to bypass integrity models by injecting malicious code. A way to prevent this would be continuous monitoring to detect and stop unauthorized modifications.