In the first assignment, I created a basic reference monitor vulnerable to various attacks, which could effectively negate its effectiveness. Subsequently, I attempted to validate the reference monitor using several attack scenarios, during which I discovered vulnerabilities that could undermine the security layer. The following is an identification of these vulnerabilities and the methods I employed to rectify them.

The *openfile* Function

- **Filename Validation**: I have restricted the allowed characters to ensure filenames contain only lowercase letters, digits, or periods. However, the check for invalid characters has been commented out because of the *CodeUnsafeError* exception, which may mean the check is unnecessary.
- **Error Handling**: Enhanced error handling to catch *FileNotFoundError* and *FileInUseError*, raising exceptions to prevent unauthorized or invalid access.

The *readat* Function

- **EOF and Negative Offset Checks**: Implemented checks to prevent reading past the file's end and using negative offsets. The *readat* Function initially allowed reading beyond the end of the file marker and did not handle negative offsets, which could lead to data leakage or crashes.
- **Concurrency Control**: A lock is included to manage concurrent access, maintaining the integrity of file reads across multiple threads.

The *writeat* Function

- **Pending Write Management**: I have ensured that pending writes are committed before a new write is processed.
- **File Size Consistency**: The file size is now correctly updated after each write. It reflects the actual size and prevents writes beyond the current size using *the pending_new_size* variable.

The *undo* Function

- **Transactional Integrity**: By introducing the undo Function, I've added a mechanism to reverse undo(uncommitted) writes, thereby maintaining transactional integrity.
- **Size Reversion**: If an undo operation is called, the file size is reverted to reflect the last committed state, ensuring that any subsequent write operations are accurately checked against the correct file size using *last_committed_size*.
- **Thread Lock**: The addition of thread locks within the *undo* function prevents race conditions, allowing safe reversals even in a multi-threaded environment.

The *close* Function

- **Final Write Commit**: Pending writes are committed before the file is closed, which ensures no data is left unwritten, or it will potentially lead to data loss if not doing so.
- **Redundant Close Attempts**: Added checks to prevent closing an already closed file, which could otherwise raise exceptions or lead to undefined behavior.