Assignment 2 Part 3

Reference monitor: `reference_monitor_sr6895.r2py`

Vulnerabilities covered in Part1

I used locks to prevent multiple threads running on the same file to access close, writeat, undo at the same time. This prevented any unexpected behaviour for attackcases with multiple threads.
I also added the lock.release() in finally blocks to make sure the lock is released if an exception is thrown in the try block.
I made sure RepyArgumentError and SeekPastEndOfFileError was thrown for offsets to writeat which were < 0 or past the lenght of file. This was done by keeping track of the length of the file.

Vulnerabilities corrected for Part3

Vulnerability

I had missed throwing FileClosedError when writeat was called after closing a file. This resulted in the wrong Error or no error being thrown for a writeat after closing file.

Resolution

I added a variable (self.closed) that kept track whether the close() function had been called.
The variable gets assigned to False when the file is opened i.e. inside ___init___() function
If close() had been called the variable will be set to True.
In `check_offset()` function if the offset is not negative, then the variable for closed file will be checked and FileClosedError will be thrown if the variable is set to True.
RepyArgumentError is checked first because Repy also throws RepyArgumentError before FileClosedError.

PFB the snippets

```
def __init__(self, filename, create):
    ...
    self.closed = False
    ...

def check_offset(self, offset):
    len_data = self.length
    if self.pending_data is not None and self.pending_offset is not None:
        len_data = len(self.pending_data)+self.pending_offset
        if  len_data < self.length:
            len_data = self.length
    if offset < 0:
        raise RepyArgumentError("Min value is 0.")
    if (self.closed):
        raise FileClosedError("File '" + self.filename + "' is already closed!")
    if offset > len_data:
        raise SeekPastEndOfFileError("Seek offset extends past the EOF!")

def close(self):
    self.lock.acquire(True)
    try:
        if self.pending_data is not None and self.pending_offset is not None:
            self.LPfile.writeat(self.pending_data, self.pending_offset)
        self.LPfile.close()
        self.closed = True
    finally:
        self.lock.release()
```