My first vulnerability was not checking if a file can be written to after it has been closed. The attack file tries to exploit the behavior of my class by attempting to write to a closed file. The expected behavior is that when trying to write to a closed file, my LPFile class should raise an exception, FileClosedError.
When the file is closed (self.is_open is False), my writeat method should be able to catch the FileClosedError exception and simply pass it without re-raising or propagating the exception. In other words, even if the exception is raised within the security layer, it will be caught and suppressed.
My approach is to propagate an exception when attempting to write to a closed file.

```
if not self.is_open:
    raise FileClosedError("The file is already closed.")
```

My second vulnerability was not paying attention to the issue of offsets. The attack file tries to exploit the behavior of my class by attempting to write to a file at a negative offset. The expected behavior is that when trying to write at a negative offset, my LPFile class should raise an exception, RepyArgumentError.
I've patched this vulnerability by adding a condition.

```
if offset < 0:
    raise RepyArgumentError("Negative offset is not allowed.")
```

My third vulnerability was that I did not check whether the offset exceeds the current length of the file. The attack file tries to exploit the behavior of my class by attempting to write to a position that exceeds the file length. The expected behavior is that when trying to write to a position beyond the file length, my LPFile class should raise an exception, SeekPastEndOfFileError.
First, I get the current length of the file and check if the current offset is greater than the file length. If it is, an exception should be raised.

```
file_length = len(self.LPfile.readat(None, 0))
if offset > file_length:
    raise SeekPastEndOfFileError("The offset is beyond the end of the file.")
```