```
Exploit 1:
The first exploit I intend to use is a buffer overflow in the writeLog() function. The tempString array in this
function is only
1024 bytes long, but as long as there are no end-of-file or newline characters input by the user, the loop that
writes each character
input to the array will not terminate, allowing us to overwrite memory beyond the bounds of the array. This can be
exploited to
overwrite the return address of the writeLog() function.
If the return address is set to the tempString array, it is possible to first write shellcode that launches an
admin shell to the array,
then overwrite the return address of the function to the address of that array. Then when the function returns,
the shellcode at that
address will be executed, resulting in a shell.

Exploit 2:
The second exploit also involves a buffer overflow, but it does not use this to overwrite the function return
address. In the copyFile() function, there is a call
to execlp() made that uses the variables user and dst as two of its arguments. dst has to be a valid file name in
order for the function to not return before the execlp()
call is made, but user is a 16-byte character array that is not checked or overwritten before the execlp() call is
made. Using the call to strcpy() on line 142, it is
possible to overwrite local variables until user, which can be set to whatever we want. To get a shell using this,
the arg buffer should be formatted with the
first 72 bytes being a valid file name, and the next 72 + 4 + 8 + 8 + 72 bytes being any data (except the null
character).
The next 16 bytes (the user variable) should be the string ";/bin/sh;\0" where \0 is a single null character. This
will let the strcpy() on line 142 write a valid file name to dst, then
overwrite the src, chmod, sourcefile, destfile, tempString, and then user variables. Later on, a call to execlp()
is made with the variable user as one of its arguments.

Exploit 3:
This exploit uses the privledges of the program to rewrite the sudoers file with an entry that gives the student
user sudoer privledges. From there, a simple sudo -s can be used to get a root shell.
As the is_blocked() function checks the absolute path of the file to be checked in/out versus the list of
restricted files/directories, if you try to check in a file named block.list, it will pass the
is_blocked() test, becasue the path would be /opt/bcvs/block.list (for example) compared to /opt/bcvs/.bcvs/-
block.list, the entry in the block list intended to stop the block list itself from being
touched. As the function uses strstr() for the comparison, which looks for substrings of a string in another
string, the program will think that the file is not blocked.
Later on in the code, (if the passed-in file path is not blocked), the program will call fopen(dst, "w"), where
dst is a path to /opt/bcvs/.bcvs/%passed-in file name%. The "w" option in fopen()
will create a new file with that name, or if there is already a file named that in the directory, it will replace
it with a blank file. This means that it's possible to
call ./bcvs ci block.list and have the program overwrite its own block.list file with a blank one.
After destroying the block list, you can check in a custom sudoers file (where the student account has full
access), create a symbolic link to /etc/sudoers with the same name as this new sudoers file, and then have bcvs
checkout the custom sudoers file. It will follow the symlink, fopen() the real sudoers file, and write the fake
sudoers file to it.
```