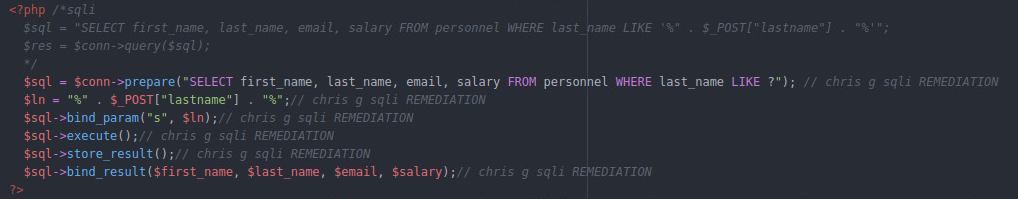
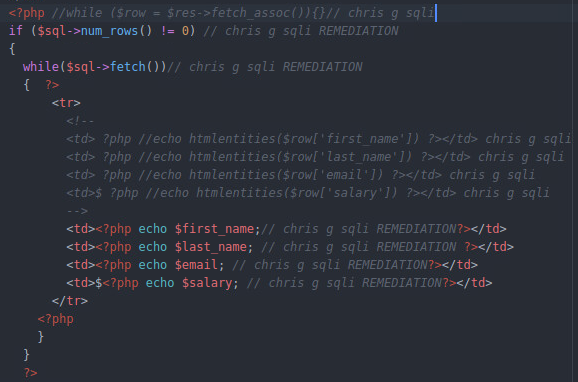
**Personnel Files App**

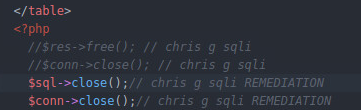
***In all screencaps, the vulnerable code can be seen commented out above or to the side***

**XSS:**

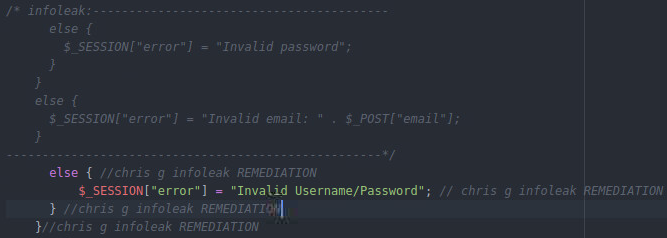
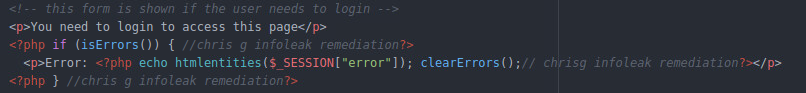
** This vulnerability printed untrusted input directly to the screen without any kind of checks or filtration to prevent html script tags from being read from the input field, and consequently printed and executed on the client machine. Using the htmlentities() function, we escaped the dangerous html characters which prevent the contents of a script tag from executing.   
*(name at end of comment: chris g xss)*

**SQLI:**

This vulnerability allowed the client to abuse the system and escape the string, inserting a logical tautology which would return the contents of the database, or another SQL appropriate statement which would reveal information about the structure of the database. The solution is to use prepared statements, concatenating the user’s search statement using a builtin function which escapes dangerous characters.



**Infoleak:**

 The fact that different errors are displayed on the login screen depending on if the email is correct or not reveals valid emails/usernames in the db even if the passwords are incorrect (Also: a change was made in index.php to display the session error on the login page, as it was only displaying errors on the personnell search page, preventing the login errors from being displayed in the first place.)  
change to index.php:

**Cookie Order Program**

***In all screencaps, the vulnerable code can be seen commented out above or to the side***

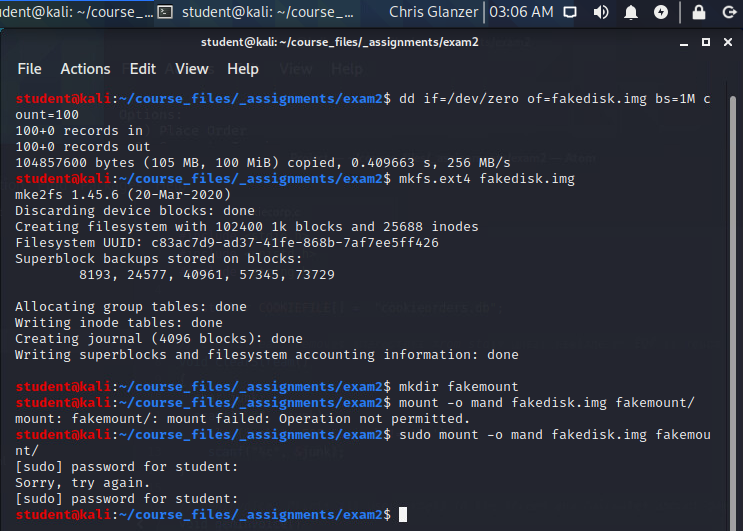
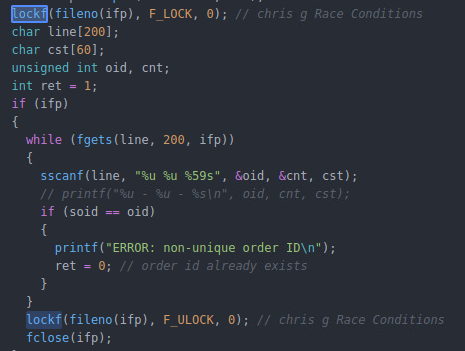
**Buffer Overflow:**

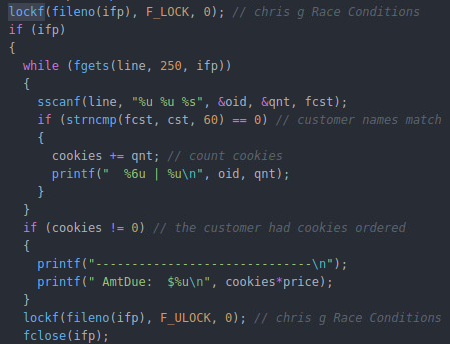
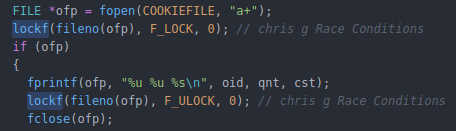
The function gets() does not stop reading input until it encounters a newline or EOF without regard for the size of the buffer, allowing one to write beyond the boundaries of the buffer. The solution is to use the alternative function fgets, which takes the buffer size as an argument.

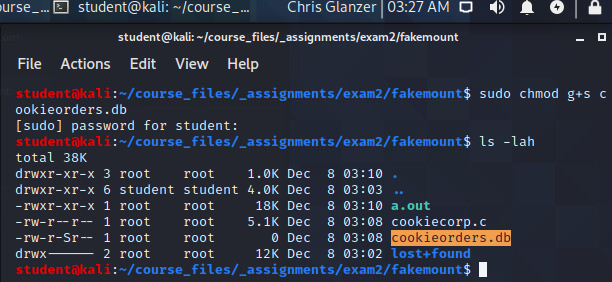


**Race Condition:**

The race condition results from a lack of locks on the database file, allowing multiple programs to open and operate on it at the same time. The solution is to do the following:

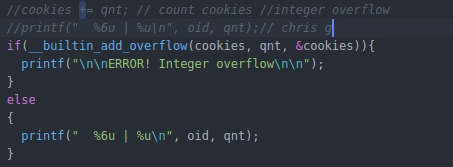
1. Put the db file on a file system that makes use of mandatory locks
2. make use of the lockf function, which prevents other programs from operating on a file while you maintain a lock.

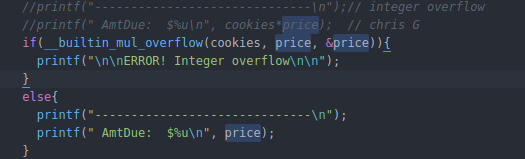


1. Set the set user group bit using chmod g+s so that your locks are actually maintained 

**Integer Overflows:**

Integer overflows are the result of unchecked mathematical operations resulting in incorrect values being presented or stored, and the solution is to check all calculations that could potentially overflow using a builtin function, and only run the process if it returns . In the first example below, the variable ‘cookies’ acts as an accumulator for the sum of all cookies on their account to date, which may overflow if they have ordered an ungodly amount of cookies in various non-offending transactions. In the second, the direct multiplication of two user provided values may exceed the maximum value for their type(unsigned int).





**User Controlled Format String:**

**** Printing a user provided string directly as a variable is dangerous because they can enter their own null format string using the %n format specifier (enabling them to overwrite memory),