Noel O'Kane

15076083

Assess security principles in a software application

CS2S562

Contents

[Report on Program 006.Zip 2](#_Toc5394983)

[Integer Vulnerabilities 2](#_Toc5394984)

[String Vulnerabilities 2](#_Toc5394985)

[Memory Vulnerabilities 3](#_Toc5394986)

[Formatted IO Vulnerabilities 3](#_Toc5394987)

[File IO Vulnerabilities 4](#_Toc5394988)

[Pointer Vulnerabilities 4](#_Toc5394989)

[Automated quality tool usage 5](#_Toc5394990)

[Secure Pattern 1 6](#_Toc5394991)

[Secure Pattern 2 6](#_Toc5394992)

[Summary and overall verdict 7](#_Toc5394993)

[Report on Program 032.Zip 8](#_Toc5394994)

[Integer Vulnerabilities 8](#_Toc5394995)

[String Vulnerabilities 8](#_Toc5394996)

[Memory Vulnerabilities 9](#_Toc5394997)

[Formatted IO Vulnerabilities 9](#_Toc5394998)

[File IO Vulnerabilities 10](#_Toc5394999)

[Pointer Vulnerabilities 10](#_Toc5395000)

[Secure Pattern 1 12](#_Toc5395001)

[Secure Pattern 2 12](#_Toc5395002)

[Summary and overall verdict 13](#_Toc5395003)

[Report on Program 036.Zip 14](#_Toc5395004)

[Integer Vulnerabilities 14](#_Toc5395005)

[String Vulnerabilities 14](#_Toc5395006)

[Memory Vulnerabilities 15](#_Toc5395007)

[Formatted IO Vulnerabilities 15](#_Toc5395008)

[File IO Vulnerabilities 16](#_Toc5395009)

[Pointer Vulnerabilities 16](#_Toc5395010)

[Secure Pattern 1 18](#_Toc5395011)

[Secure Pattern 2 18](#_Toc5395012)

[Summary and overall verdict 19](#_Toc5395013)

# Report on Program 006.Zip

## Integer Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Unsigned integer wrap | 1. Use of C99 types to safely handle sizing 2. When performing mathematical operations, is there checks that the resulting value will fit in its given variable 3. Use of size constraints like INT32\_MAX | 1. The inttypes.h library included but not used 🗷 2. Checks made when generating a random number and calculating distance 🗹 3. Also used when checking random number and calculating distance🗹 |
| Signed Integer overflow | 1. When performing mathematical operations, is there checks that the resulting value will fit in its given variable 2. Operators << and >> have not been used | 1. Checks made when generating a random number and calculating distance 🗹 2. Operators have not been used🗹 |
| Loss of data due to type casting | 1. Ensure that numbers being casted can fit in their variables | 1. When calculating Fahrenheit code attempts to multiply 1.8 to and int (converting double to int) resulting in loss of data🗷 |

Overall integer security performance verdict (0 = poor, 5 = excellent): **2.5**

## String Vulnerabilities

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerability Type | Checks Made | | Result |
| Must have enough storage for data and null terminator | 1. Check for dynamic memory allocation to help prevent overflow 2. Use of strings whenever possible | 1. User input is allocated dynamically🗹 2. Char arrays avoided when possible🗹 | |
| Not creating a std::string from a null pointer | 1. Checks made for if string can be nullptr | | 1. User input is checked even though it could never be nullptr🗹 |
| Use valid references, pointers and iterators to reference elements of basic\_string | 1. Check that any iteration using the standard template library is done with appropriate checks | | 1. Program has no use of string iterators🗹 |

Overall string security performance verdict (0 = poor, 5 = excellent): **4**

## Memory Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Accessing freed memory | 1. Check that any memory items that have been deleted have not been accessed again. | 1. 3 pointers used which are all deleted just before the program terminates with no further references to them 🗹 |
| Has only freed memory allocated dynamically | 1. Check that anything declared dynamically is deleted | 1. All pointers are deleted after their use is no longer needed 🗹 |
| Has allocated sufficient memory | 1. Use of C99 types for pointers to allow consistent variable size 2. Check for use of sizeof() when running checks on size of pointers. | 1. No use of C99 types 🗷 2. No use of sizeof() when checking pointers but good use of sizeof() when reading and writing to file. 🗹 |

Overall memory management performance verdict (0 = poor, 5 = excellent): **4.5**

## Formatted IO Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Dealing with unformatted I/O | 1. Check for use of canonicalization (converting raw data that’s use could be ambiguous ) 2. Check for use of normalisation (stripping unneeded parts from input) 3. Check for use of validation to check that data is an expected input 4. Check for use of output sanitisation (removing of sensitive data after it is no longer needed) | 1. No use of canonicalization 🗷 2. Uses serpercision() to round the decimal number to 2 decimal places🗹 3. Validation checking for if invalid numbers are entered but not for when incorrect strings are entered 🗷 4. No use of sanitisation to remove sensitive data 🗷 |
| Variadic functions | 1. Check that each use of the % formatter has a corresponding parameter | 1. No use of Variadic functions 🗷 |
| Dealing with GUI input | 1. Check that any GUI validation is handled (textbox expecting a number should not accept characters) | 1. Program is not a GUI. 🗷 |

Overall formatted IO performance verdict (0 = poor, 5 = excellent): **0.5**

## File IO Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Avoid TOCTOU (Time Of Check, Time Of Use) race conditions while accessing files | 1. Checks that file can be accessed before trying to read or write to it. 2. Check that more than 1 process isn’t trying to access the stream at any time. | 1. Error message displayed when there is an issue opening file to read/write 🗹 2. Stream is closed when not in use and is only used by 1 process at a time 🗹 |
| Distinguishing between characters read from a file and EOF | 1. Check that the appropriate checks have been made to test for end of file ( using methods such as : feof and ferror) | 1. No checks made to check for end of file🗷 |
| Do not assume that fgets() returns a nonempty string when successful | 1. Check that program checks for \n and replaces it to not return an empty string when reading from a file | 1. No use of fgets()🗹 |

Overall file IO performance verdict (0 = poor, 5 = excellent): **2.5**

## Pointer Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| NULL pointer | 1. Check that all pointers are pointing to something when declared. 2. Check that any functions that could potentially return NULL to a pointer are handled | 1. Pointers all declared correctly 🗹 2. No pointers are returned from functions🗷 |
| Confusing pointer and value | 1. Check that any pointers passed to other functions are using the correct syntax | Correct syntax used when passing pointer values and pointer addresses used 🗹 |
| Point miscalculated | 1. Check that any calculation using pointers is referenced correctly. | No use of arithmetic used on pointers 🗷 |

Overall pointer performance verdict (0 = poor, 5 = excellent): **2**

## Automated quality tool usage

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Compiler errors | 1. Checking if there are any build errors. 2. Checking if there are any build warnings | 1. No build errors 2. 4 warnings present |
| Use of testing support libraries | 1. Check if any libraries have been included to test the code | 1. There is no use of external libraries to help test code |
| Hidden warnings | 1. Check if all rules are set | 1. Rules set to default. When switched over to all rules there are 59 warnings present |

Overall automated quality tool usage verdict (0 = poor, 5 = excellent): **2**

## Secure Pattern 1

|  |  |  |
| --- | --- | --- |
| Design Pattern | Checks Made | Result |
| Authorization Pattern | Check resource independence | The authorization pattern is constructed in a way that it is not dependant on any other area of the program. Therefore, the pattern could in theory be copied for use in another program |
| Different types of users | There is support for more than 1 user, the functionality of different types of users is flawed |
| Authorisation rules | As there is no different types of uses there is no rules for who can do what. |

Overall pattern verdict (0 = poor, 5 = excellent): **2**

## Secure Pattern 2

|  |  |  |
| --- | --- | --- |
| Design Pattern | Checks Made | Result |
| Authenticator Pattern | Check resource independence | The authenticator pattern is constructed in a way that it is not dependant on any other area of the program. Therefore, the pattern could in theory be copied for use in another program |
| Check for correct user information | Program denies access if incorrect login info has been entered |
| Use of software token | There is functionality for a software token by using the ProofOfID class |

Overall pattern verdict (0 = poor, 5 = excellent): **5**

### Summary and overall verdict

|  |  |  |
| --- | --- | --- |
| **Check Type** | **Comment** | **Verdict** |
| Integer | Good check on arithmetic operations but no use of C99 types. Missed opportunity with the warning for converting a double to integer | 2.5 |
| String | Very good use of strings | 4 |
| Memory | Memory is handled well ensuring to delete objects when they are finished with | 4.5 |
| Formatted I/O | The only formatting done is setting the precision to 2 decimal places | 0.5 |
| File I/O | Not much security when it comes to reading the end of a file | 2.5 |
| Pointers | Use of pointers in this program was minimal by the programmer | 2 |
| Tool Usage | There is little use of automated tools to test code | 2 |
| Authenticator Pattern | Good resource independence and authorization rules but some weaknesses in the types of users (only one type possible) | 2 |
| Authorisation Pattern | Very good use of the Authenticator pattern. Token is implemented well | 5 |
| **OVERALL** | **A reasonably secure program** | **25/45** |

# Report on Program 032.Zip

## Integer Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Unsigned integer wrap | 1. Use of C99 types to safely handle sizing 2. When performing mathematical operations, is there checks that the resulting value will fit in its given variable 3. Use of size constraints like INT32\_MAX | 1. Very good use of C99 types for integers and doubles🗹 2. All mathematical operations for unsigned integers are separated into functions of the SecureUInt32 class. 🗹 3. There is good use of INT32\_MAX and UINT32\_MAX🗹 |
| Signed Integer overflow | 1. When performing mathematical operations, is there checks that the resulting value will fit in its given variable 2. Operators << and >> have not been used | 1. All mathematical operations for signed integers are separated into functions of the SecureInt32 class. 🗹 2. There is no use of << or >> 🗹 |
| Loss of data due to type casting | 1. Ensure that numbers being casted can fit in their variables | 1. All type casting is handled safely 🗹 |

Overall integer security performance verdict (0 = poor, 5 = excellent): **5**

## String Vulnerabilities

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerability Type | Checks Made | Result | |
| Must have enough storage for data and null terminator | 1. Check for dynamic memory allocation to help prevent overflow 2. Use of strings whenever possible | | 1. Checks made using secure string class. 🗹 2. Use of char arrays handled very well in the secure string class🗹 |
| Not creating a std::string from a null pointer | 1. Checks made for if string can be nullptr | 1. No instances found where string is created from nullptr🗹 | |
| Use valid references, pointers and iterators to reference elements of basic\_string | 1. Check that any iteration using the standard template library is done with appropriate checks | 1. Basic\_string not used but appropriate checks have been made with the secure string class🗹 | |

Overall string security performance verdict (0 = poor, 5 = excellent): **5**

## Memory Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Accessing freed memory | 1. Check that any memory items that have been deleted have not been accessed again. | 1. SecureString class has a clear method to safely delete memory before reassignment🗹 |
| Has only freed memory allocated dynamically | 1. Check that anything declared dynamically is deleted | 1. No incorrect freeing of memory found🗹 |
| Has allocated sufficient memory | 1. Use of C99 types for pointers to allow consistent variable size 2. Check for use of sizeof() when running checks on size of pointers. | 1. Program uses custom types handled very well to ensure there is no issues with sizing of variables🗹 2. No use of sizeof to check size of pointers🗷 |

Overall memory management performance verdict (0 = poor, 5 = excellent): **4.5**

## Formatted IO Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Dealing with unformatted I/O | 1. Check for use of canonicalization (converting raw data that’s use could be ambiguous ) 2. Check for use of normalisation (stripping unneeded parts from input) 3. Check for use of validation to check that data is an expected input 4. Check for use of output sanitisation (removing of sensitive data after it is not longer needed) | 1. No evidence of canonicalization🗷 2. Using .width when taking user entry to take only the first character from the input🗹 3. Switch statements used so only the expected selections will return true🗹 4. When the user enters their username and password the data is safely used and deleted 🗹 |
| Variadic functions | 1. Check that each use of the % formatter has a corresponding parameter | 1. No instances of variadic functions🗹 |
| Dealing with GUI input | 1. Check that any GUI validation is handled (textbox expecting a number should not accept characters) | 1. The only textbox entry in the program is typing user name and password which is compared against the encrypted text document storing the login data 🗹 |

Overall formatted IO performance verdict (0 = poor, 5 = excellent): 4.5

## File IO Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Avoid TOCTOU (Time Of Check, Time Of Use) race conditions while accessing files | 1. Checks that file can be accessed before trying to read or write to it. 2. Check that more than 1 process isn’t trying to access the stream at any time. | 1. Checks have been made in the authentication pattern when reading from the database of users for if the file can’t be opened🗹 2. Use of the stream is handled safely to open and close in the SettingsView class🗹 |
| Distinguishing between characters read from a file and EOF | 1. Check that the appropriate checks have been made to test for end of file ( using methods such as : feof and ferror) | 1. No checks found to confirm end of file has been reached🗷 |
| Do not assume that fgets() returns a nonempty string when successful | 1. Check that program checks for \n and replaces it to not return an empty string when reading from a file | 1. No use of fgets evident in program🗹 |

Overall file IO performance verdict (0 = poor, 5 = excellent): **3.5**

## Pointer Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| NULL pointer | 1. Check that all pointers are pointing to something when declared. 2. Check that any functions that could potentially return NULL to a pointer are handled | 1. No uninitialized pointers declared🗹 2. No function can return null pointer🗹 |
| Confusing pointer and value | 1. Check that any pointers passed to other functions are using the correct syntax | 1. All use of pointers are using the correct syntax 🗹 |
| Point miscalculated | 1. Check that any calculation using pointers is referenced correctly. | 1. No use of pointer calculation found🗹 |

Overall pointer performance verdict (0 = poor, 5 = excellent): **5**

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Compiler errors | Checking if there are any build errors.  Checking if there are any build warnings | No build errors  No build warnings |
| Use of testing support libraries | Check if any libraries have been included to test the code | There is extensive use of different testing libraries |
| Hidden warnings | Check if all rules are set | When all rules set only 1 warning is present |

Overall automated quality tool usage verdict (0 = poor, 5 = excellent): **5**

## Secure Pattern 1

|  |  |  |
| --- | --- | --- |
| Design Pattern | Checks Made | Result |
| Authorization Pattern | Check resource independence | The authorization pattern is constructed in a way that it is not quite independent but could be quite simply adjusted to be able to use in a different program |
| Different types of users | There is very good support for muti-level access control. This incudes functionality for top level users change permissions for other users. The database for the users is stored in an “triple encrypted” text file. |
| Authorisation rules | There is good implementation for different rules bases on level of user. This includes only top level users can set the satellite to self destruct |

Overall pattern verdict (0 = poor, 5 = excellent): **5**

## Secure Pattern 2

|  |  |  |
| --- | --- | --- |
| Design Pattern | Checks Made | Result |
| Secure Logger | Check resource independence | The logger pattern is constructed in a way that it is independent of the program and could be re-used elsewhere. The program also then used the pattern to log its own sensor data |
| Check that there is error handling for if the log file cannot be accessed | Errors are handled very well if the program has problems accessing the log files. |
| Check if multiple ways of logging to a file has been implemented | User can choose to store the logs in a binary or text file. You can then choose to store as plain text or encrypted |

Overall pattern verdict (0 = poor, 5 = excellent): **5**

### Summary and overall verdict

|  |  |  |
| --- | --- | --- |
| **Check Type** | **Comment** | **Verdict** |
| Integer | Very good integer security. Functions for lots of different arithmetic options | 5 |
| String | Use of safe string class was very good. | 5 |
| Memory | Use of memory management was very efficient | 4.5 |
| Formatted I/O | Very good checks for user input as well as the safe deletion of sensitive username and password | 4.5 |
| File I/O | Good use of file IO but missed opportunities when checking for end of file | 3.5 |
| Pointers | Programmer clearly has a very good understanding of the use of pointers | 5 |
| Tool Usage | Very complex testing used as well as use of 3rd party software to test code, evidenced in design document | 5 |
| Authenticator Pattern | Good resource independence and authorization rules but some weaknesses in the types of users (only one type possible) | 5 |
| Secure logger | The logger pattern is implemented very well and is adapted well by the program for its uses | 5 |
| **OVERALL** | **A very impressive program. The programmer has a clear understanding of secure programming** | **42.5/45** |

# Report on Program 036.Zip

## Integer Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Unsigned integer wrap | 1. Use of C99 types to safely handle sizing 2. When performing mathematical operations, is there checks that the resulting value will fit in its given variable 3. Use of size constraints like INT32\_MAX | 1. No use of C99 types🗷 2. No checks made when handling mathematical operation for unsigned integers🗷 3. No use of size constraints 🗷 |
| Signed Integer overflow | 1. When performing mathematical operations, is there checks that the resulting value will fit in its given variable 2. Operators << and >> have not been used | 1. No checks made when handling mathematical operations for signed integers🗷 2. No << or >> operators use 🗹 |
| Loss of data due to type casting | 1. Ensure that numbers being casted can fit in their variables | 1. No casting used in program🗹 |

Overall integer security performance verdict (0 = poor, 5 = excellent):**1**

## String Vulnerabilities

|  |  |  |  |
| --- | --- | --- | --- |
| Vulnerability Type | Checks Made | Result | |
| Must have enough storage for data and null terminator | 1. Check for dynamic memory allocation to help prevent overflow 2. Use of strings whenever possible | | 1. No checks made to prevent string overflow🗷 2. No char arrays used🗹 |
| Not creating a std::string from a null pointer | 1. Checks made for if string can be nullptr | 1. No checks made to test if string is nullptr🗷 | |
| Use valid references, pointers and iterators to reference elements of basic\_string | 1. Check that any iteration using the standard template library is done with appropriate checks | 1. No incorrect references or iterators found🗹 | |

Overall string security performance verdict (0 = poor, 5 = excellent): **0.5**

## Memory Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Accessing freed memory | 1. Check that any memory items that have been deleted have not been accessed again. | 1. Objects are deleted when the program is about to terminate, therefore are not accessed again. 🗹 |
| Has only freed memory allocated dynamically | 1. Check that anything declared dynamically is deleted | 1. No dynamically declared items found🗷 |
| Has allocated sufficient memory | 1. Use of C99 types for pointers to allow consistent variable size 2. Check for use of sizeof() when running checks on size of pointers. | 1. No use of C99 types in program🗷 2. No use of sizeof() to compare size of pointers🗷 |

Overall memory management performance verdict (0 = poor, 5 = excellent): 1

## Formatted IO Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Dealing with unformatted I/O | 1. Check for use of canonicalization (converting raw data that’s use could be ambiguous ) 2. Check for use of normalisation (stripping unneeded parts from input) 3. Check for use of validation to check that data is an expected input 4. Check for use of output sanitisation (removing of sensitive data after it is not longer needed) | 1. No use of canonicalization detected🗷 2. Use of width() when using cin to restrict the length of the string that will be accepted🗹 3. When setting temperature the program only accepts values from 0-30 🗹 4. No removal of sensitive data from the program when it isn’t being used🗷 |
| Variadic functions | 1. Check that each use of the % formatter has a corresponding parameter | 1. No use of variadic functions 🗹 |
| Dealing with GUI input | 1. Check that any GUI validation is handled (textbox expecting a number should not accept characters) | 1. Program not a GUI 🗹 |

Overall formatted IO performance verdict (0 = poor, 5 = excellent): **1**

## File IO Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Avoid TOCTOU (Time Of Check, Time Of Use) race conditions while accessing files | 1. Checks that file can be accessed before trying to read or write to it. 2. Check that more than 1 process isn’t trying to access the stream at any time. | 1. Checks are there for if a file fails to open when logging data from sensors🗹 2. No checks made to stop more than one process from access to stream🗷 |
| Distinguishing between characters read from a file and EOF | 1. Check that the appropriate checks have been made to test for end of file ( using methods such as : feof and ferror) | 1. No checks have been made to confirm if end of file has been reached🗷 |
| Do not assume that fgets() returns a nonempty string when successful | 1. Check that program checks for \n and replaces it to not return an empty string when reading from a file | 1. No use of fgets🗷 |

Overall file IO performance verdict (0 = poor, 5 = excellent): **0.5**

## Pointer Vulnerabilities

|  |  |  |
| --- | --- | --- |
| Vulnerability Type1 | Checks Made | Result |
| NULL pointer | 1. Check that all pointers are pointing to something when declared. 2. Check that any functions that could potentially return NULL to a pointer are handled | 1. No uninitialized pointers when declared 2. No checks made for if functions can return null to a pointer |
| Confusing pointer and value | 1. Check that any pointers passed to other functions are using the correct syntax | 1. No instances of confuction for the correct syntax of pointers |
| Point miscalculated | 1. Check that any calculation using pointers is referenced correctly. | 1. No pointer calculation present |

Overall pointer performance verdict (0 = poor, 5 = excellent): **2**

|  |  |  |
| --- | --- | --- |
| Vulnerability Type | Checks Made | Result |
| Compiler errors | Checking if there are any build errors.  Checking if there are any build warnings | No build errors  No build warnings |
| Use of testing support libraries | Check if any libraries have been included to test the code | There is no use of external libraries to help test code |
| Hidden warnings | Check if all rules are set | When all rules set 89 warnings are present |

Overall automated quality tool usage verdict (0 = poor, 5 = excellent): **1**

## Secure Pattern 1

|  |  |  |
| --- | --- | --- |
| Design Pattern | Checks Made | Result |
| Authorization Pattern | Check resource independence | While there has been an attempt to independently implement the authenticator there are bugs which would |
| Different types of users | There is support for more than 1 user, the functionality of different types of users is flawed |
| Authorisation rules | As there is no different types of uses there is no rules for who can do what. |

Overall pattern verdict (0 = poor, 5 = excellent): **1**

## Secure Pattern 2

|  |  |  |
| --- | --- | --- |
| Design Pattern | Checks Made | Result |
| Secure Logger | Check resource independence | The logger pattern is constructed in a way that it is independent of the program and could be re-used elsewhere. |
| Check that there is error handling for if the log file cannot be accessed | Errors are handled well if the program has problems accessing the log files. |
| Check if multiple ways of logging to a file has been implemented | User can choose to store the logs in a binary, text file on log to console. You can then choose to store as plain text, encrypted or html. |

Overall pattern verdict (0 = poor, 5 = excellent): **3**

### Summary and overall verdict

|  |  |  |
| --- | --- | --- |
| **Check Type** | **Comment** | **Verdict** |
| Integer | Poor attempt at integer security | 1 |
| String | Strings are not handled securely at all | 0.5 |
| Memory | Little effort made to control memory management in this program | 1 |
| Formatted I/O | There is no checking of input from the user. Only check made is that new temperature can be between 0 and 30 | 1 |
| File I/O | There is no security when it comes to input and output of files. An attacker could quite easily penetrate this program | 0.5 |
| Pointers | There aren’t really any errors with pointer but there is very little use of pointers outside the design patterns. | 2 |
| Tool Usage | Very little tool usage and a considerable amount of errors | 1 |
| Authorization Pattern | Good resource independence and authorization rules but some weaknesses in the types of users (only one type possible) | 1 |
| Secure logger | Logger implemented well although the program doent actually make use of the logger to regularly record system infomation | 3 |
| **OVERALL** | **A reasonably secure program** | **11/45** |