Dennis Persaud

CS-405-H2379 Secure Coding

Southern New Hampshire University

20EW2

Audit Report

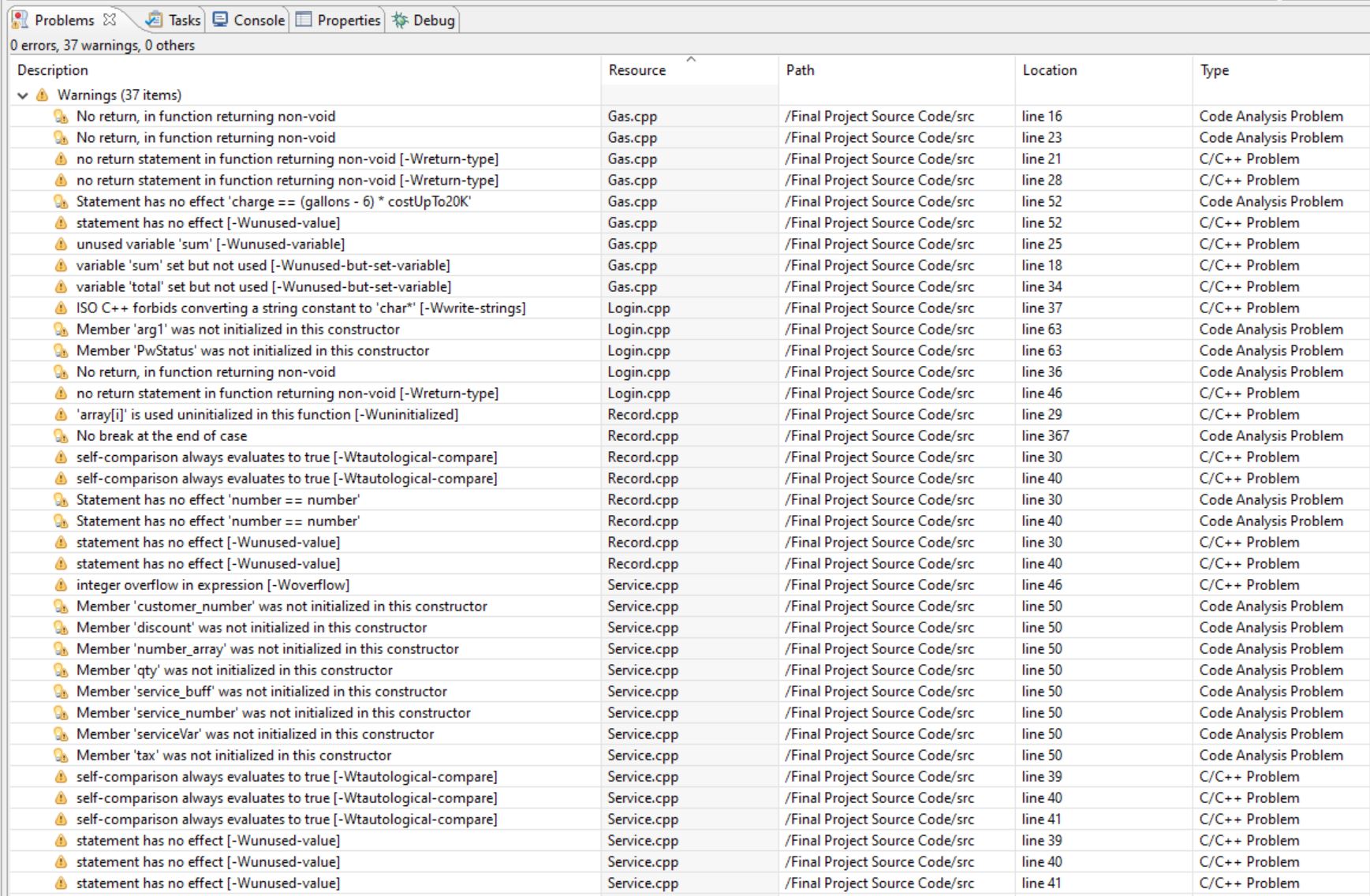
**Summary of Methods**

1. Describe how and when you used methods to **visually inspects** the code to identify the vulnerabilities you found.

When visually inspecting the code, I first checked where the developer accepted user input in each unit of code. I found that the developer had accepted user input in multiple units through out different class files. I then checked to verify if the developer created any functionality to validate the user input in each unit where the user input was accepted. I found that the developer did not validate the user input anywhere in the code, this indicates to me that this code may potentially contain vulnerabilities and bugs.

1. Describe how and when you utilized **compiler errors** and/or warnings to identify the vulnerabilities that you found.

When trying to compile the code I received 37 warnings but the code compiled successfully with 0 errors. This confirmed my suspicion from my visual inspection that the code contained potentially contained vulnerabilities and bugs. The warnings that were thrown by the compiler indicated that there had been bugs present in several files contained within this project. Those files are Gas.cpp, Login.cpp, Record.cpp, and Service.cpp. The compiler warnings and their description can be found in the attached image below.



1. Explain how and when you used a **static analysis tool** to identify the vulnerabilities you found.

After reviewing the warnings thrown by the compiler, I then used cppchecker to perform a static analysis of the code. After running the static code analysis, I found several errors, warnings, and style warnings contained within different class files of this program. The static code analysis indicated that there are more severe errors contained within the code in comparison to the warnings thrown by the compiler. The resources that were found to contain these errors within the project are Gas.cpp, Login.cpp, Record.cpp, Service.cpp, and Service.h. A summary of each of the errors and warnings found by the static code analysis tool is shown in the image below.

****

**Vulnerability Findings**

**Gas.cpp**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Code Analysis Problem | Source File:  Final Project Source Code/src/Gas.cpp  Line Number: 16 – 22 | Compiler Warning | The function on line 16 – 22 is missing a return statement. | **int** **Gas::gasChargeCalc**(**int** a, **int** b)  {  **int** sum;  sum = a + b;  **return** sum;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Code Analysis Problem | Source File:  Final Project Source Code/src/Gas.cpp  Line Number: 23 – 28 | Compiler Warning | The function on line 23 – 28 is missing a return statement and the operation was commented out. | **double** **Gas::gasChargeCalc**(**int** a, **double** b)  {  **double** sum;  sum = a + b;  **return** sum;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Code Analysis Problem | Source File:  Final Project Source Code/src/Gas.cpp  Line Number: 31 – 64 | Compiler Warning & cppchecker | The function on line 31 – 64 contains variables that are not initialized properly. The “total” variable is not used in the output on line 62, and the statement on line 52 is using the incorrect assignment operator. The condition on line 50 was an always true condition do to the less than equal symbol being used. | **void** **Gas::gasFeeCalculation**()  {  **double** gallons = 0.0, charge = 0.0, total = 0.0;  **const** **int** fee = 15;  **double** costUpTo6K = 2.35,  costUpTo20K = 3.75,  costOver20K = 6.00;  **system**("cls");  cout << "\n\n\n\tEnter the total number of gallons used, divided by 1000: ";  cin >> gallons;  **if** (gallons > 20){  charge = (gallons - 20) \* costOver20K;  charge = charge + (14 \* costUpTo20K);  charge = charge + (6 \* costUpTo6K);  }  **else** **if** (gallons > 6 && gallons < 20){  charge = (gallons - 6) \* costUpTo20K;  charge = charge + (6 \* costUpTo6K);  }  **else**{  charge = gallons \* costUpTo6K;  }  total = gasChargeCalc(charge,fee);  cout << "\n\n\n\tYou have used " << gallons << " thousand gallons of water." << **endl**;  cout << "\n\n\n\tYour total water bill is $" << setprecision(2) << fixed << total;  **getch**();  } |

**Login.cpp**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Code Analysis Problem | Source File:  Final Project Source Code/src/Login.cpp  Line Number: 39 – 49 | Compiler Warning | In the function on line 39 – 49 the “arg1” variable was not properly initialized, and the function was missing a return statement. | **int** **Login::testUsernamePassword**(**void**) {  **char** arg1[] = "Test Username and Password.";  **while** (arg1[i] != '\0') {  buff[i] = arg1[i];  i++;  }  buff[i] = '\0';  **printf**("buff=%s\n", buff);  **return** 0;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Login.cpp  Line Number: 27 – 37 | Cppchecker | In the function on line 27 – 37, the variables were not properly initialized before the strcpy() function was called, resulting in a buffer overflow. | **int** **Login::copyPassword**(){  **char** a[13];  **char** b[13];  **strcpy**(a, "copypassword");  **strcpy**(b, "copypassword");  **strcpy**(c, a);  **strcat**(c, b);  **printf**("a = %s\n", a);  **return** 0;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Login.cpp  Line Number: 12 – 17 | Cppchcker | In the function on line 12 – 17 the “Password” variable was not properly initialized. The vulnerable function gets() was used instead of fgets(). | **void** **Login::enterPassword**(**void**){  **puts**("Enter 8 character password");  **char** Password[9];  **fgets**(Password, 9, stdin);  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Login.cpp  Line Number: 51 – 65 | Cppchecker | In the function on line 51 – 65 the “passwordCheck” variable was not properly initialized. The vulnerable function gets() was used instead of fgets(). | **void** **Login::isPasswordOK**(**void**) {  **puts**("Enter Password");  **char** passwordCheck[9];  **fgets**(passwordCheck, 9, stdin);  PwStatus = (**strcmp**(passwordCheck, "goodpass") == 0);  **if** (PwStatus == **false**) {  **puts**("Access Denied");  }  **else** {  **puts**("Access Granted");  }  } |

**Record.cpp**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Record.cpp  Line Number: 206 – 213 | CppChecker | In the function on line 206 – 213 the allocation and deallocation variables were mismatched. | **void** **Record::deleteServiceCharacter**(**void**)  {  **char** \*n = **new** **char**;  **char** \*m = (**char** \*)**malloc**(1);  **free**(m);  **delete** n;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Record.cpp  Line Number: 79 – 84 | Cppchecker | The memory address in the function on line 79 – 84 was not initialized with the appropriate memory address. | **void** **Record::displayNumber**()  {  **int** \*number = &NULL;  \*number = 5;  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Record.cpp  Line Number: 91 – 99 | Cppchecker | In the function on line 91 – 99 the incorrect logic was used. | **void** **Record::displayServiceItem**(**struct** service\_object \*obj)  {  **if** (obj == NULL)  {  obj->value = 2;  } **else** {  obj->value = 1;  }  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| C/C++ Problem | Source File:  Final Project Source Code/src/Record.cpp  Line Number: 158 – 186 | Cppchecker | In the function on line 158 – 186 the operation multipiled a signed integer by and unsigned integer. | **void** **Record::modifyService**()  {  **int** no,found=0;  **system**("cls");  recordMenu.serviceMenu();  cout<<"\n\nTo Modify ";  cout<<"\n\nPlease Enter The Service Number of The Service:\n";  cin>>no;  fp.open("Service.dat",ios::*in*|ios::*out*);  **while**(fp.read((**char**\*)&serv,**sizeof**(Service)) && found==0)  {  **if**(serv.retsno()==no)  {  serv.showService();  cout<<"\nPlease Enter The New Details of Service"<<**endl**;  serv.createService();//Overflow  **int** pos=-1\*(**int**)**sizeof**(serv);  fp.seekp(pos,ios::*cur*);  fp.write((**char**\*)&serv,**sizeof**(Service));  cout<<"\n\n\t Record Updated";  found=1;  }  }  fp.close();  **if**(found==0)  cout<<"\n\n Record Not Found ";  **getch**();  } |

**Part II: Case Study Analysis** Based on the provided documentation on the following two case studies from Predicting Software Assurance Using Quality and Reliability Measures, provide a brief analysis of the two case studies.

1. Case One: Database Vulnerabilities (Apple Coding Vulnerability, found on page 19) Given the particular scenario pertaining to the database system, evaluate security aspects of program designs and architectures for defending against attacks:

1. Explain the security issues present, indicating the potential risks that the issues pose. [CS-405-04]

The security issues present in Apple’s “goto fail;” vulnerability was that a duplicate line of code was present in their implementation of their SSLVerifySignedServerKeyExchange function. The second “goto fail;” line within the function gets executed regardless of the whether the predicate in the if statement is true or false. This causes the value zero to be returned to the caller, by passing the signature verification. This flaw in the program permits an attacker to use forged certificates to snoop on affected users via a man-in-the-middle attack.

1. Recommend testing types or processes necessary to identify the vulnerabilities. [CS-405-04]

This bug in Apple’s sslKeyExchange file was most likely caused by the developer copying and pasting lines of code without verifying what they pasted, and a lack of code reviews. The processes necessary to identify these vulnerabilities would be to perform more manual code reviews before the code is shipped. However, code reviews are not foolproof and does not guarantee that the reviewer will catch the error. Another way to identify a vulnerability like this would be to create automated unit tests, and implement thorough test cases.

1. Case Two: Architecture-Specific Vulnerabilities (Heartbleed Vulnerability, found on page 21) Given the particular scenario pertaining to a server, evaluate security aspects of program designs and architectures for defending against attacks:
2. Explain the security issues present, indicating the potential risks that the issues pose. [CS-405-04]

The Heartbleed Vulnerability occurred in the “assert” function of the OpenSSL toolkit. OpenSSL is an open-source implementation of the secure socket layer (SSL) and transport security layer (TLS) protocols used for securing web communications. Because OpenSSL is open-source it is used in a large amount of development projects. The vulnerability present in the “assert” function was caused by a lack of bounds checking for the length of the message being passed to it. This causes the client to retrieve additional data from the server’s memory, which could potentially contain sensitive information.

1. Recommend testing types or processes necessary to identify the vulnerabilities. [CS-405-04]

The bug was most likely caused accidentally by the developer, who may have forgot to verify the length of the message being passed to the “assert” function. Since this bug was caused by the lack of bounds checking, it would be hard to be caught using a static code analysis tool. A recommended process necessary to identify these vulnerabilities would be to perform effective peer code reviews more often. The reviewers should follow the design specification closely when verifying the code.

**References**

Synopsys Editorial Team. 2014. Understand the Apple ‘goto fail;’ vulnerability. Retrieved from [Understanding the Apple ‘goto fail;’ vulnerability | Synopsys](https://www.synopsys.com/blogs/software-security/understanding-apple-goto-fail-vulnerability-2/)

Josh Fruhlinger. 2017. What is the Heartbleed bug, how does it work and how was it fixed? Retrieved from [What is the Heartbleed bug, how does it work and how was it fixed? | CSO Online](https://www.csoonline.com/article/3223203/what-is-the-heartbleed-bug-how-does-it-work-and-how-was-it-fixed.html)

Paul Ducklin. 2014. Anatomy of a “got fail” – Apple’s SSL bug explained, plus and unofficial path of OS X! Retrieved from [Anatomy of a “goto fail” – Apple’s SSL bug explained, plus an unofficial patch for OS X! – Naked Security (sophos.com)](https://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/)