Final Project

Rebecca Glanfield

CS 405: Secure Coding

10/05/2020

## Part I: Secure Code Audit Report

**Executive Summary**

Vulnerabilities identified in the program included weaknesses related to string-formatted output, memory management, pointers and the quality of code. Among the files containing errors and warnings were the gas.cpp file, the Login.cpp, login.h, Record.cpp and Service.cpp files. Staring with the Gas.cpp file, the first warning was related to an unused variable "sum" and another "total", as well as, missing return statements. The Login.cpp file was also missing a return statement and the Login.h file had an error related to a value being assigned to a non-static data member. There were also warnings related to unused values and variables in the Record.cpp and Service.cpp files. After taking a closer look at these issues I have compiled a more detailed description of each of these weaknesses, as well as, recommendations that correct or reduce the risk of vulnerabilities.

**Summary of Methods**

The first method utilized, to determine vulnerabilities in the program, was a visual inspection of the code. I loaded the program into Eclipse and decided to look through each source file line by line to determine what errors I could detect and resolve on my own. I was also provided a Final Project Issues Checklist, that I used to guide my review. The next step was to build the project and look through the compiler errors and warnings in each file. Eclipse has a problems tab that displayed 34 errors and 21 warnings and identified the source files related to each. The last toll utilized was Cpp check. It is a static analysis tool for C and C++ code. It was installed in the Eclipse environment and was used to detect undefined behavior and dangerous coding constructs.

**Vulnerability Findings**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| String-  Formatted  Output | Source File:  Login.cpp  Line number: 28 | Static  Analysis | Buffer is  accessed out of  bounds | Highlighted code  added:  if (strlcpy(a,  “copypassword”, sizeof(a)) >=  sizeof(a))  return -1; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Memory  Management | Source File:  Record.cpp  Line number: 211 | Static  Analysis | Mismatching  allocation and  deallocation | Highlighted code  added:  free(m); |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Pointers | Source File:  Record.cpp  Line number: 28 and 37 | Static  Analysis | Array 'array [10]' accessed at index 11, which is out of bounds | Highlighted code  added:  int i = 10; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Code Quality | Source File:  Gas.cpp  Line: 16 and 25  Source File:  Gas.cpp  Line: 26  Source File:  Gas.cpp  Line Number: 55  Source File:  Login.cpp  Line number: 36 -46  Source File  Record.cpp  Line number: 30 and 40  Source File  Record.cpp  Line number: 367 | Compiler  Warning  Static  Analysis  Visual  Inspection  Static  Analysis  Visual  Inspection  Compiler  warning | No return  statement returning non-void.  Variable set but not used. this can lead to future defects an  vulnerabilities  Statement has no effect  No return, in function returning non -void  Statement has no effect  No break at the end of case. This can result in multiple sections executing rather than the one intended | Highlighted code  added:  int Gas::gasChargeCalc(int a, int b)  {  int sum;  sum = a + b;  return sum;  }  double Gas::gasChargeCalc(int a, double b)  {  double sum;  // sum = a + b;  return sum;  }  double Gas::gasChargeCalc(int a, double b)  {  double sum;  sum = a + b;  return sum;  }  charge = (gallons - 6) \* costUpTo20K;  return 0;  number = i;  Getch(); break; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weakness Category** | **Location** | **Method of Identification** | **Description** | **Remediation Recommendation** |
| Integer Arithmetic |  |  |  | Highlighted code  added: |

## Part II: Case Study Analysis

**Case One: Database Vulnerabilities**

The case study related to the Apple coding vulnerability is about a security risk that was most likely caused by the practice of cutting and pasting lines of code into the program. This led to a duplicate line of code that caused the program to bypass the block of code that verified user's credentials. According to Declan McCullagh, a Software Engineer who has worked for CNET, among other companies, the vulnerability was found in iOS and OSX software utilized by multiple Apple products and left millions of customers data vulnerable to attacks through applications like Safari, Mail, iCloud and other Apple based applications (McCullagh, D., 2014). The vulnerability would have allowed an attacker to gain access to sensitive personal data, using invalid credentials. While there are a number of techniques and recommendations for reducing vulnerabilities, the case study determined, a thorough personal review or peer review could have identified the error (Woody, C., Ellison, R., & Nichols, W., 2014). The case study released by the Software Engineering Institute ultimately concluded that this type of vulnerability was related to carelessness and a lack of adhering to coding best practices.

**Case Two: Architecture-Specific Vulnerabilities**

The case study about the Heartbleed vulnerability examined an error in Open SSL software used for securing web communications. According to the case study, the vulnerability occurred in the assert function that initiates the heart beat protocol to verify that the OpenSSL server was live.

(Woody, C., Ellison, R., & Nichols, W., 2014). The vulnerability occurred in the lack of a bounds check to verify the payload length specified was not longer than the actual data sent. The failure to verify that the input data met the specified payload resulted in the ability for attackers' access to sensitive information, affecting nearly 500, 000 secure web servers. The Cybersecurity and Infrastructure Security Agency explains the vulnerability further stating, the vulnerability," allows an attacker to retrieve private memory of an application that uses the vulnerable OpenSSL library in chunks of 64k at a time and an attacker can repeatedly leverage the vulnerability to retrieve as many 64k chunks of memory as are necessary to retrieve the intended secrets" (National Cyber Awareness System, 2014). The prevention of similar attacks starts with adhering to secure coding principles and a discipled approach to the removal of defects and other vulnerabilities.

***References***

McCullagh, Declan (February 25, 2014). Apple finally fixes 'gotofail' OS X security hole.Retrieved from: https://www.cnet.com/news/apple-finally-fixes-gotofail-os-x-security-hole/

National Cyber Awareness System (April 8, 2014). *Alert (TA14-098A).*

## *OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160)*. Retrieved from: https://us- cert.cisa.gov/ncas/alerts/TA14-098A

Woody, C., Ellison, R., & Nichols, W. (2014). *Predicting Software Assurance Using Quality and Reliability Measures*.