Employee Salary Calculator Program and Analysis

Raneem Kazma

Colorado State University Global Campus

My code receives 3 double values from a user to be used in a salary calculation. Additionally, my code has 3 pointers for all 3 double variables. The vulnerabilities in my code can be caused by integer and integer formatting, the use of pointers and integer formatting. Integer vulnerabilities include integer signedness, integer overflow, widthness issue and data type issues. Pointers issues include dangling, void and null pointers.

Integers vulnerabilities may cause unexpected behavior or program termination; however, they are not easily exploitable. Integer signedness can occur if a declared signed integer was passed to function expecting an unsigned integer. In a scenario where a negative number is passed as a maximum size argument and later used as an unsigned value, a heap overflow may ensue. Integer overflows occur when an integer is too large for its data type (Younan Y., 2004). To avoid signedness errors, a programmer must look out for the following; signed integers used in comparisons, unsigned integers used in arithmetic operations and unsigned integers being compared to unsigned integers (Basic Integer Overflows). Integer overflow are dangerous as they cannot be detected by the compiler. The program therefore cannot assess whether the correctness of a calculation. However, integer overflow cannot be exploited as they do not overwrite memory (Basic Integer Overflows). If an integer is unable to hold an integer due to its size, the number is reduced to modulo the maximum number an integer can hold plus 1. This in turn truncates the number and discards the portion of the number the integer cannot hold (Basic Integer Overflows). Additionally, a widthness issue may occur when a larger integer is assigned to a location with a smaller size (Coker Z.).

Formatting vulnerabilities can exist when an input to format functions is executed as a command. Format string function usually require a format specifies, which specifies how the output will appear. When a format specifier requires an argument, it expects to find it on the stack. A vulnerability can occur if attackers can specify a format string to a format function, granting them control to pop from the stack. One way to mitigate this vulnerability is to compare the type of arguments passed to a function and the type of arguments a function expects (Younan Y., 2004). Format string attacks can also be used to exploit variadic functions. Variadic functions accept a variable number of arguments based on their implementation. Variadic functions are not type safe and a mismatch between the argument types can be exploited by an attacker to tamper with memory (Biswas P., 2017).

References

Coker, Z., Hasan, S, Overbey, J., Hafiz, M. & Kastner, C. (n.d.) *Integers in C: An open invitation to security attacks?.*

Younan, Y., Joosen, W., & Piessens, F. (2004). *Code Injection in C and C : A Survey of*

*Vulnerabilities and Countermeasures*. Belgium: Department of Computer Science, K.U.Leuven.

Basic Integer Overflows. (n.d.). Retrieved from

<https://www.cs.utexas.edu/~shmat/courses/cs380s_fall09/blexim.txt>

Biswas, P., Federico, A. D., Carr, S. A., Rajasekaran, P., Volckaert, S., Franz, M., &

Payer, M. (2017, January 1). Venerable Variadic Vulnerabilities Vanquished. Retrieved from https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/biswas

#include <iostream>

#include <iomanip>

using namespace std;

int main(int argc, const char \* argv[]) {

double hours = 0;

double rate = 0;

double overtime = 0;

double overtimeRate = 1.5;

double salary = 0;

cout << "Enter your hours worked toaday" << endl;

cin >> hours;

double\* ptr1 = new double(hours);

cout << "Enter your hourly rate" << endl;

cin >> rate;

double\* ptr2 = new double(rate);

cout << "Enter your over time hours" << endl;

cin >> overtime;

double\* ptr3 = new double(overtime);

cout << "Your hours worked taday are " << \*ptr1 << endl;

cout << "Your hours worked taday are stored at " << ptr1 << endl;

cout << "Your hourly rate is " << \*ptr2 << endl;

cout << "Your hourly rate is stored at " << ptr2 <<endl;

cout << "Your over time hours are " << \*ptr3 <<endl;

cout << "Your over time hours are stored at " << ptr3 <<endl;

salary = (hours\*rate) + (overtime\*overtimeRate\*rate);

cout << "Your salary for today is ";

cout << salary << setw(10)<<setprecision(5) << endl;

delete ptr1;

delete ptr2;

delete ptr3;

return 0;

}

A screenshot of a cell phone

Description automatically generated