Paradigms HW2

1. Explain how recursion can be inefficient, and how tail-call optimization solves it.
   1. Recursion can be inefficient due to the amount of memory it can take up. When a function is called recursively, it adds additional memory in the form of stack frames onto the call stack. With large enough recursive calls, this can use lots of memory or even cause a stack overflow if the additional memory exceeds the limit allocated to the stack. Tail-call optimization solves this by limiting the additional stack frames added onto the call stack. This is because when reaching the “tail” case of a recursive call (last return before getting a result), it jumps to back to the first recursive call, not adding onto the stack.
2. Analyze this code. What is its type? What does it do? Does it consider every case? Explain why such code is not inefficient when written in Haskell.
   1. The code type is integers, specifically it takes 2 Ints and returns a list of Integers.
   2. This code returns a list of all consecutive numbers from x to y, increasing. For example, if x is 3 and y is 8, it will return [3,4,5,6,7,8].
   3. Yes, this code considers every case. The first case is if x > y, then it will return an empty list because there are no increasing numbers from x to y. The second case is if x == y, which will return a list of just the element y. The last case is when x < y, and it will return all the integers between x and y, inclusive.
   4. While this code is not tail-call optimized, Haskell uses lazy evaluation, meaning it will only generate the list as it is needed, meaning it won’t use up unnecessary memory for large recursive calls.

sumByKey :: [(Int, Int)] -> Either String (Int, Int)

sumByKey pairs = sumByKeyTail pairs (0, 0) where --call tail recursive function

sumByKeyTail :: [(Int, Int)] -> (Int, Int) -> Either String (Int, Int) --input is a list of tuples, output is either a tuple or an error message

sumByKeyTail [] acc = Right acc --base case when list is empty, return accumulator value

sumByKeyTail ((key, value):xs) (sum0, sum1)

| key == 0 = sumByKeyTail xs (sum0 + value, sum1) --if the key is a 0, add the value to the current total of sum0

| key == 1 = sumByKeyTail xs (sum0, sum1 + value) --if the key is 1, add value to the current total of sum1

| otherwise = Left $ "Invalid key: " ++ show key --if the key is not 0/1, then there is an invalid key, display the incorrect value

sumPairs :: [Int] -> [Int]

sumPairs [] = [] --Case where input list is empty

sumPairs [x] = [x] --Case where there is only one element in the list

sumPairs (x:y:restOfList) = (x + y) : sumPairs restOfList --General case, sums first two elements, and recursively calls with rest of list

type Student = (Int, Int, Int, Int) --(exams, hw, projects, quizzes)

studentAverage :: Student -> Double --function that gives weighted grade for each student

studentAverage (ex, hw, pr, qu) =

fromIntegral (ex \* 30 + hw \* 30 + pr \* 30 + qu \* 10) / 100

classAverages :: [Student] -> [Double]

classAverages = map studentAverage --map student average function to each student in the class

removeVowels :: String -> String

removeVowels = filter (\char -> notElem char "AEIOUaeiou") -- use elem function to check if current character is in a list of vowels

reversedWords :: String -> [String]

reversedWords = map reverse . words -- . used to compose the functions map reverse and words, the words function splits input into separate words, and the map reverse applies reverse to each separate word.

1. What is “Use After Free”? What are the two common causes?
   1. Use After Free occurs when trying to reference memory that has already been freed up, which can cause a program to crash, use unintended values, or execute code. The two common causes are error conditions and other exceptional circumstances and confusion over which part of a program is responsible for freeing memory.
2. What is “Improper Input Validation”? How is it related to CWE-89 and CWE-78?
   1. Improper Input Validation is when a program receives an input or data but doesn’t validate it correctly and makes sure that it’s safe. This can be used by attackers to create inputs that can cause changes in control flow, code execution, etc. It is similar to CWE-89 and 78 in that they are the result of mishandling inputs, causing vulnerabilities in a system.
3. What is “Integer Overflow or Wraparound”. How can it affect security?
   1. Integer overflow or wraparound occurs when a calculation happens, and an integer is incremented to a value that is higher than what can be stored. This may cause the calculation to “wrap around” and become a negative or much smaller integer than anticipated. This can affect security when the wrap around is unintended, and especially when it is due to a user’s inputs. For example, user inputs causing wraparounds could be used to control loops (infinite looping), determining things such as memory allocation, data corruption, etc.
4. Why is “Use of Hard-coded Credentials” considered a weakness?
   1. Use of hard-coded credentials can be exploited in cases where users or attackers have access to the source code where the credentials are located, giving them potential access to things such as accounts, keys, etc., when it wasn’t intended for them to have access to them.
5. “Out-of-bounds Write” and “Out-of-bounds Read” are more specific versions of what common weakness?
   1. Improper restriction of operations within the bounds of a memory buffer.
6. What is the “Uncontrolled Resource Consumption” weakness? What are the two common situations where incorrect implantation can lead to “Uncontrolled Resource Consumption”?
   1. The uncontrolled resource consumption refers to when a program fails to properly manage the allocation and maintenance of limited resources (includes memory, file system storage, CPU, and others). Two common situations where incorrect implantation can lead to this CWE are the lack of throttling for number of allocated resources and losing all references to a resource before reaching shutdown stage.
7. What is a “NULL Pointer Dereference”. Which major languages allow for programs that allow this weakness?
   1. A NULL pointer dereference is when an application dereferences a pointer that has an expected valid value but is actually NULL, usually causing crashes or program exits. The major language that allows this weakness is C/C++.
8. What is Cross-site scripting? Briefly describe the three main types.
   1. Cross-site scripting is when a program doesn’t properly neutralize user input before it is sent to output on a web page that is served to other users. The three main types are:
      1. Reflected XSS – when a server reads data directly from an HTTP request and sends it to the HTTP response. This sent data can be exploited by attackers who use it to send malicious content to users, where it will execute on their browser.
      2. Stored XSS – when a database for information stores dangerous data, it can cause damage when read into dynamic content by the program.
      3. Type 0: DOM-based XSS – a type of attack where there is vulnerability in a client-side scripting environment that incorrectly validates it, allowing it be executed in the web page.
9. What is the “Path Traversal” weakness? Briefly describe two ways it can be taken advantage of?
   1. Path traversal weakness occurs when a program uses external input to allow access to directories/folders that a user shouldn’t have access to. Two ways it can be taken advantage of are:
      1. Using special elements such as ../ allowing for attackers to escape outside the intended restricted location.
      2. Injecting null byte (0 or NUL) can allow attackers to truncate a filename in order to give wider access to the attack.
10. Consider CWE-200, “Exposure of Sensitive Information to an Unauthorized Actor”. Briefly explain Example 1 given on its webpage.
    1. The program is a login authentication, giving different output based on if the user gave either the wrong username or password. This can be exploited by attackers because it gives them information on whether they are entering the wrong password or username, allowing them to better guess values.
11. What is CWE-352, “Cross-Site Request Forgery (CSRF)”? Give an example.
    1. Cross-site request forgery occurs when a web application does not properly validate if a request was purposely sent by the actual user who submitted it. An example of this would be a user using an online banking website that saves login information. An attacker sends a link to a malicious website that opens the banking website and is able to make transfers because the banking website saved the login info, all done in the background without the user knowing.
12. Describe CWE-434, “Unrestricted Upload of File with Dangerous Type”. Give a real world example of software that avoids this weakness. What could happen, in this example, if the software did not avoid this weakness?
    1. Unrestricted upload of file with a dangerous type occurs when a program does not properly restrict files that a user can upload, allowing them to upload files that can be processed within a program’s environment. An example of software that avoids this weakness is Google Drive, which restricts the types of files you can upload to it. Without this, users could upload and spread malware much easier through Google Drive.
13. Which CWE does it have? What problems could this cause? Discuss how this program can be fixed so that it no longer has this weakness.
    1. This is an example of deserialization of untrusted data. This could allow attackers to modify deserialized data/code because it was not properly verified. It can also give attackers the ability to perform malicious actions, including remote code execution, generating a shell, etc. This program can be fixed by validating the input, sText, before it is deserialized, making sure that it can be trusted.