Anh Tran
pat200000
Section 501

ASSIGNMENT 2

Chapter 15: Functional Programming Languages

1)

- Recursion can be inefficient as it has more space requirement as it keeps adding function into the stack memory until the base case is reached. Recursion also takes more time to finish because of the function calls and returns overhead. Infinite recursive calls may occur due to some mistake in specifying the base condition, which on never becoming false, keeps calling the function, which may lead to a system CPU crash.
- Tail recursion works off the realization that some recursive calls don't need to "continue from where they left off" once the recursive call returns. Specifically, when the recursive call is the last statement that would be executed in the current context. Tail recursion is an optimization that doesn't bother to push a stack frame onto the call stack in these cases, which allows your recursive calls to go very deep in the call stack.

2)

The type of this code is is (Ord t, Num t) => t -> t -> [t] as the function takes in 2 numbers which are of type Num and they have to be ordered also to compare them and return a list with the number of elements in that list depends on how far the numbers are apart from each other using the following conditions:

- If x > y then it returns an empty list
- If x == y then it returns a list with y as its only element
- Otherwise it calls a recursive loop while incrementing the value of x until one of the other two outcomes happen. Therefore slowly adding each increment into the list.

3) Solution in source code file sumByKey.hs

```
sumByKey :: [(Int, Int)] -> (Int, Int)
sumByKey pairs = sumByKeyHelper pairs (0, 0)
  where
    sumByKeyHelper [] acc = acc
    sumByKeyHelper ((k,v):xs) (sum0, sum1)
      | k == 0 = sumByKeyHelper xs (sum0 + v, sum1)
      | k == 1 = sumByKeyHelper xs (sum0, sum1 + v)
      | otherwise = error "Invalid key found. Only 0 or 1 are
allowed."
```

```
main :: IO()
main = print (sumByKey [(1,2),(1,2),(1,2)])
```

4) Solution in source code file sumPairs.hs

```
sumPairs :: [Int] -> [Int]
sumPairs [] = []
sumPairs [x] = [x]
sumPairs (k:v:t) = (k + v) : sumPairs t

main :: IO()
main = print (sumPairs [1,2,3,4,5])
```

5) Solution in source code file classAverage.hs

```
-- Exams: 30%; Homework: 30%; Projects: 30%; Quizzes: 10%
classAverages :: Fractional b => [(b, b, b, b)] -> [b]
classAverages students = zipWith (*) weights (map (/ fromIntegral
(length students)) (classTotals students))
    where
        weights = [0.3,0.3,0.3,0.1]
        classTotals :: (Num a) => [(a, a, a, a)] -> [a]
        classTotals students = classAveragesHelper students [0, 0,
0, 0]
                where
                classAveragesHelper [] acc = acc
                classAveragesHelper ((a,b,c,d):xs) [exams, homework,
projects, quizzes]
                    = classAveragesHelper xs [exams + a, homework +
b, projects + c, quizzes + d]

main :: IO()
main = print (classAverages [
    (80.4,79.8,89.3,90.5),
    (87.4,65.2,74.6,76.1),
    (81.4,75.2,94.6,86.1),
    (67.4,55.2,74.6,86.1)])
```

6) Solution in source code file removeVowel.hs

```
removeVowel :: String -> String
removeVowel "" = ""
```

```
removeVowel str = filter (`notElem` "aeiouAEIOU") str

main :: IO()
main = print (removeVowel "Hello World")
```

7) Solution in source code file reversedWord.hs

```
reverseWord :: [a] -> [a]
reverseWord [] = []
reverseWord [x] = [x]
reverseWord (x:xs) = reverseWord xs ++ [x]

main :: IO()
main = print (reverseWord "Hello World!")
```

Common Weakness Enumeration

8) Use-After-Free (UAF) is a vulnerability related to incorrect use of dynamic memory during program operation. If after freeing a memory location, a program does not clear the pointer to that memory, an attacker can use the error to hack the program.

   Use-after-free errors have two common and sometimes overlapping causes:
   - Error conditions and other exceptional circumstances.
   - Confusion over which part of the program is responsible for freeing the memory.

9) Improper input validationor unchecked user input is a type of vulnerability in computer software that may be used for security exploits. This vulnerability is caused when "the product does not validate or incorrectly validates input that can affect the control flow or data flow of a program."

   Examples include:
   - Buffer overflow
   - Cross-site scripting
   - Directory traversal
   - Null byte injection
   - SQL injection
   - Uncontrolled format string

10) An integer overflow or wraparound occurs when an integer value is incremented to a value that is too large to store in the associated representation. When this occurs, the value may wrap to become a very small or negative number.
    While this may be intended behavior in circumstances that rely on wrapping, it can have security consequences if the wrap is unexpected. This is especially the case if the integer overflow can be triggered using user-supplied inputs. This becomes

security-critical when the result is used to control looping, make a security decision, or determine the offset or size in behaviors such as memory allocation, copying, concatenation, etc.

11) CWE-798: Use of Hard-coded Credentials
The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for its own inbound authentication, outbound communication to external components, or encryption of internal data.

Hard-coded credentials typically create a significant hole that allows an attacker to bypass the authentication that has been configured by the software administrator. This hole might be difficult for the system administrator to detect. Even if detected, it can be difficult to fix, so the administrator may be forced into disabling the product entirely.

There are two main variations:
Inbound: the software contains an authentication mechanism that checks the input credentials against a hard-coded set of credentials.
Outbound: the software connects to another system or component, and it contains hard-coded credentials for connecting to that component.

In the Inbound variant, a default administration account is created, and a simple password is hard-coded into the product and associated with that account. This hard-coded password is the same for each installation of the product, and it usually cannot be changed or disabled by system administrators without manually modifying the program, or otherwise patching the software. If the password is ever discovered or published (a common occurrence on the Internet), then anybody with knowledge of this password can access the product. Finally, since all installations of the software will have the same password, even across different organizations, this enables massive attacks such as worms to take place

The Outbound variant applies to front-end systems that authenticate with a back-end service. The back-end service may require a fixed password which can be easily discovered. The programmer may simply hard-code those back-end credentials into the front-end software. Any user of that program may be able to extract the password. Client-side systems with hard-coded passwords pose even more of a threat, since the extraction of a password from a binary is usually very simple.

12) CWE-787: Out-of-bounds Write : (Common Weakness Enumeration)

- The software writes data past the end, or before the beginning, of the intended buffer.
- Typically, this can result in corruption of data, a crash, or code execution. The software may modify an index or perform pointer arithmetic that references a

memory location that is outside of the boundaries of the buffer. A subsequent write operation then produces undefined or unexpected results.

CWE-125: Out-of-bounds Read : (Common Weakness Enumeration)

- The software reads data past the end, or before the beginning, of the intended buffer.
- Typically, this can allow attackers to read sensitive information from other memory locations or cause a crash. A crash can occur when the code reads a variable amount of data and assumes that a sentinel exists to stop the read operation, such as a NUL in a string. The expected sentinel might not be located in the out-of-bounds memory, causing excessive data to be read, leading to a segmentation fault or a buffer overflow. The software may modify an index or perform pointer arithmetic that references a memory location that is outside of the boundaries of the buffer. A subsequent read operation then produces undefined or unexpected results.

13) The software does not properly control the allocation and maintenance of a limited resource, thereby enabling an actor to influence the amount of resources consumed, eventually leading to the exhaustion of available resources.

Limited resources include memory, file system storage, database connection pool entries, and CPU. If an attacker can trigger the allocation of these limited resources, but the number or size of the resources is not controlled, then the attacker could cause a denial of service that consumes all available resources. This would prevent valid users from accessing the software, and it could potentially have an impact on the surrounding environment. For example, a memory exhaustion attack against an application could slow down the application as well as its host operating system.

There are at least three distinct scenarios which can commonly lead to resource exhaustion:

- Lack of throttling for the number of allocated resources
- Losing all references to a resource before reaching the shutdown stage
- Not closing/returning a resource after processing

Resource exhaustion problems are often result due to an incorrect implementation of the following situations:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for releasing the resource

14) NULL Pointer Dereference weakness occurs where software dereferences a pointer with a value of NULL instead of a valid address.

NULL pointer dereference erros are common in C/C++ languages. Pointer is a programming language data type that references a location in memory. Once the value of the location is obtained by the pointer, this pointer is considered dereferenced. The NULL pointer dereference weakness occurs where application dereferences a pointer that is expected to be a valid address but instead is equal to NULL

15) Cross-site scripting is a security vulnerability that occurs when an attacker injects malicious scripts into web applications that are later executed by other users' browsers. The three main types of XSS are:
    - Stored XSS: Malicious scripts are permanently stored on the target server (e.g., in a database) and served to users who view a specific page, leading to persistent attacks.
    - Reflected XSS: Malicious scripts are embedded in URLs or input fields, and the server reflects them back to the user's browser, executing the script when the user visits the URL or submits the form.
    - DOM-based XSS: The vulnerability arises in the Document Object Model (DOM) of a web page, where client-side scripts manipulate the DOM to execute malicious code.

16) Path Traversal is a vulnerability where an attacker can manipulate file paths to gain unauthorized access to files or directories outside of the intended directory

    Two ways it can be exploited:
    - Directory Listing: Attackers can use path traversal to access directories and list the contents, potentially exposing sensitive information.
    - File Inclusion: By manipulating file paths, attackers can include arbitrary files, such as configuration files or scripts, leading to remote code execution or information disclosure

17) Example 1 on the CWE-200 webpage demonstrates a vulnerability where a web application displays sensitive user data, such as a user's account details or personal information, to an unauthorized user without proper authentication or authorization. This exposure of sensitive information could lead to privacy breaches and identity theft.

18) Cross-Site Request Forgery is an attack where an attacker tricks a user into executing unwanted actions on a web application in which the user is authenticated. For example, an attacker could create a malicious link that, when clicked by a victim, performs actions on the victim's behalf without their consent, like changing their password.

19) CWE-434 (Unrestricted Upload of File with Dangerous Type) is a Common Weakness Enumeration entry that addresses a security vulnerability associated with the unrestricted uploading of files in web applications. This weakness can lead to severe security issues if not properly mitigated. Here are more details about CWE-434:

Description:

- CWE-434 refers to a situation where a web application allows users to upload files without imposing sufficient controls and validation on the uploaded content.
- The weakness specifically focuses on the type of files uploaded. It occurs when an application permits the upload of files with dangerous or unexpected file types.
- Attackers can exploit this weakness by uploading malicious files that appear benign (e.g., executable scripts or malware) but are disguised as legitimate files (e.g., images, documents).

Risk:

- Unrestricted file uploads can lead to various security risks, including remote code execution, data breaches, and unauthorized access.
- Attackers may upload malicious files that, when executed, compromise the server's security or allow them to gain unauthorized access.
- For example, an attacker could upload a PHP script disguised as an image file. If the server does not validate the file type and content, it may execute the script, potentially leading to a compromise of the server.

Mitigation: To mitigate CWE-434, web applications should implement the following security measures:

- File Type Validation: Check the file type by examining the file's content or using file extensions. Do not rely solely on the file extension, as it can be easily manipulated by an attacker.
- Content Validation: Analyze the content of uploaded files to ensure they do not contain malicious code or scripts.
- File Size Limit: Enforce limits on the size of uploaded files to prevent denial-of-service attacks.
- Secure File Storage: Store uploaded files in a secure location with restricted access to prevent unauthorized retrieval or execution.
- Use of Content-Disposition Header: Set the Content-Disposition header to prevent browsers from executing certain file types in response to user interactions.
- Sanitization: Use content sanitization techniques to remove or neutralize any potentially harmful elements from the uploaded files.

By implementing these security measures, web applications can reduce the risk associated with unrestricted file uploads and protect against CWE-434 vulnerabilities.

20) The main security concern in this code is the potential for a Remote Code Execution vulnerability, which could lead to arbitrary code execution on the server-side if an attacker can manipulate the input data. Common Weakness Enumeration (CWE): The CWE that relates to this issue is CWE-94: Improper Control of Generation of Code ('Code Injection').

Problem:

- No Input Validation: The code does not seem to perform proper validation or sanitization of the input received through the socket. If an attacker sends malicious input, they might be able to inject arbitrary code that gets executed when the deserialize and doWork functions are called.
- Lack of Output Sanitization: The code does not show any evidence of output sanitization before sending the result over the socket. Depending on the nature of the result, sending it without proper encoding or validation could lead to issues.

Solution:

- Input Validation: Before passing the input data (sText) to the deserialize function, you should implement thorough input validation. Validate the input against an expected format or structure and reject any input that doesn't conform to it. You can use a serialization library that provides a safe deserialization mechanism to avoid code injection.
- Output Sanitization: Ensure that the result you're sending over the socket is properly encoded or serialized to prevent unintended behavior. If the result contains any sensitive information or executable code, make sure it's appropriately handled before sending