Reviewing the code for security vulnerabilities and providing recommendations for secure coding practices. Using manual code review tools.

Vlnerable sample code for review.

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <cstdlib>

void readInput(char* buffer) {
    std::cout << "Enter your name: ";
    std::cin >> buffer; // Potential buffer overflow
}

void processFile(const char* filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cout << "Error opening file\n";
        return;
    }
    char line[100];
    while (file >> line) { // Unsafe, may overflow
        std::cout << line << std::endl;
    }
}

void executeCommand(const char* userInput) {
    char command[100];
    strcpy(command, "ls ");         // Unsafe
    strcat(command, userInput);     // Vulnerable to command injection
    system(command);                // Dangerous function
}

int* getArray() {
    int* arr = new int[10];
    return arr; // Memory not freed anywhere — potential leak
}

int main() {
    char name[20];
    readInput(name);                // Vulnerable input
    std::cout << "Hello, " << name << std::endl;

    processFile("example.txt");

    char userCommand[50];
    std::cout << "Enter command args for ls: ";
    std::cin >> userCommand;
```

```
    executeCommand(userCommand);

    int* array = getArray();
    array[11] = 42; // Out-of-bounds write
    return 0;
}
```

Manual Code Review — Vulnerabilities & Recommendations

Line/Function	Vulnerability	Recommendation
std::cin >> buffer	Buffer overflow	Use std::cin.getline(buffer, size) or std::string
while (file >> line)	>> reads a word and stores it in line without bounds checking	std::getline(file, std::string)
file >> line	Buffer overflow in file reading	Use file.getline(line, sizeof(line)) or std::string
strcpy, strcat, system	Command injection	Avoid system(), or sanitize/validate inputs carefully
new int[10] without delete[]	Memory leak	Use std::unique_ptr<int[]> or free the memory
array[11] = 42	Out-of-bounds write	Validate indices; use std::vector or bounds-checked access
char command[100]	Buffer overflow	Use std::string or safe concatenation (snprintf)

Secure Coding Practices.

- **Avoid unsafe functions:** Replace strcpy, sprintf, gets, etc., with safer alternatives like strncpy, snprintf, fgets.
- **Validate all input:** Always check user inputs for format, length, and type before using.
- **Check all return values:** File I/O, memory allocations, and system calls must be validated.
- **Use RAII and smart pointers:** Prevent memory leaks and dangling pointers using std::unique_ptr, std::shared_ptr, etc.
- **Avoid system() calls:** If absolutely necessary, sanitize all arguments thoroughly to prevent command injection.
- **Initialize all variables:** Uninitialized variables can lead to undefined behavior.
- **Use bounds-checked containers:** Prefer std::vector::at() over unchecked indexing ([]).
- **Thread safety:** When using multithreading, guard shared resources with mutexes.

Secure Code Recommendations

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <memory>
void readInput(std::string& buffer) {
```

```
    std::cout << "Enter your name: ";
    std::getline(std::cin, buffer);
}
void processFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file) {
        std::cerr << "Error opening file\n";
        return;
    }

    std::string line;
    while (std::getline(file, line)) {
        std::cout << line << std::endl;
    }
}
void executeCommandSafely(const std::string& userInput) {
    // Very basic input validation
    for (char c : userInput) {
        if (!isalnum(c) && c != '-' && c != '_') {
            std::cerr << "Invalid input\n";
            return;
        }
    }

    std::string command = "ls " + userInput;
    // Still not ideal — better to use native APIs, but okay with proper validation
    system(command.c_str());
}
std::vector<int> getArray() {
    return std::vector<int>(10, 0); // Safe, RAII-managed
}
int main() {
    std::string name;
    readInput(name);
    std::cout << "Hello, " << name << std::endl;

    processFile("example.txt");

    std::string userCommand;
    std::cout << "Enter command args for ls: ";
    std::cin >> userCommand;
    executeCommandSafely(userCommand);

    auto array = getArray();
    if (array.size() > 11) {
        array[11] = 42; // Now safe
    }

    return 0;
}
```