Problem Statement 1

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LINE\_LENGTH 256

#define MAX\_USERS 100

typedef struct {

char username[50];

char password[50];

} UserCredential;

void evaluatePasswordStrength(const char \*password, int \*strength) {

int length = strlen(password);

int hasUpper = 0, hasLower = 0, hasDigit = 0, hasSpecial = 0;

if (length < 8) {

\*strength = 0; // Weak

return;

}

for (int i = 0; i < length; i++) {

if (isupper(password[i])) hasUpper = 1;

else if (islower(password[i])) hasLower = 1;

else if (isdigit(password[i])) hasDigit = 1;

else hasSpecial = 1;

}

if (hasUpper && hasLower && hasDigit && hasSpecial) {

\*strength = 3; // Strong

} else if ((hasUpper || hasLower) && (hasDigit || hasSpecial)) {

\*strength = 2; // Medium

} else {

\*strength = 1; // Weak

}

}

void identifyVulnerabilities(UserCredential users[], int userCount) {

printf("\nPotential Vulnerabilities:\n");

for (int i = 0; i < userCount; i++) {

if (strlen(users[i].password) < 8) {

printf("User '%s' has a weak password (less than 8 characters).\n", users[i].username);

}

if (strstr(users[i].password, "password") != NULL) {

printf("User '%s' has a password that contains the word 'password'.\n", users[i].username);

}

}

}

void evaluateCompromisedPasswords(UserCredential users[], int userCount) {

printf("\nCompromised Usernames and Passwords:\n");

for (int i = 0; i < userCount; i++) {

int strength;

evaluatePasswordStrength(users[i].password, &strength);

printf("Username: %s, Password: %s, Strength: ", users[i].username, users[i].password);

switch (strength) {

case 3: printf("Strong\n"); break;

case 2: printf("Medium\n"); break;

case 1: printf("Weak\n"); break;

default: printf("Very Weak\n"); break;

}

}

}

int main() {

FILE \*file = fopen("credentials.txt", "r");

if (!file) {

perror("Failed to open file");

return EXIT\_FAILURE;

}

UserCredential users[MAX\_USERS];

int userCount = 0;

while (fgets(users[userCount].username, sizeof(users[userCount].username), file) != NULL) {

char \*token = strtok(users[userCount].username, ":");

if (token != NULL) {

strcpy(users[userCount].username, token);

token = strtok(NULL, "\n");

if (token != NULL) {

strcpy(users[userCount].password, token);

userCount++;

}

}

}

fclose(file);

evaluateCompromisedPasswords(users, userCount);

identifyVulnerabilities(users, userCount);

return EXIT\_SUCCESS;

}

Problem Statement 2:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LINE\_LENGTH 256

#define MAX\_USERS 100

#define MAX\_DEVICES 100

#define MAX\_EMAILS 10

typedef struct {

char username[50];

char password[50];

char businessInfo[100];

} UserCredential;

typedef struct {

char device[50];

char vulnerability[100];

} NetworkDevice;

void evaluatePasswordStrength(const char \*password, int \*strength) {

int length = strlen(password);

int hasUpper = 0, hasLower = 0, hasDigit = 0, hasSpecial = 0;

if (length < 8) {

\*strength = 0; // Weak

return;

}

for (int i = 0; i < length; i++) {

if (isupper(password[i])) hasUpper = 1;

else if (islower(password[i])) hasLower = 1;

else if (isdigit(password[i])) hasDigit = 1;

else hasSpecial = 1;

}

if (hasUpper && hasLower && hasDigit && hasSpecial) {

\*strength = 3; // Strong

} else if ((hasUpper || hasLower) && (hasDigit || hasSpecial)) {

\*strength = 2; // Medium

} else {

\*strength = 1; // Weak

}

}

void identifyPasswordVulnerabilities(UserCredential users[], int userCount) {

printf("\nPotential Vulnerabilities in Password Storage:\n");

for (int i = 0; i < userCount; i++) {

if (strlen(users[i].password) < 8) {

printf("User '%s' has a weak password (less than 8 characters).\n", users[i].username);

}

if (strstr(users[i].password, "password") != NULL) {

printf("User '%s' has a password that contains the word 'password'.\n", users[i].username);

}

}

}

void evaluateCompromisedPasswords(UserCredential users[], int userCount) {

printf("\nCompromised Usernames, Passwords, and Business Info:\n");

for (int i = 0; i < userCount; i++) {

int strength;

evaluatePasswordStrength(users[i].password, &strength);

printf("Username: %s, Password: %s, Business Info: %s, Strength: ",

users[i].username, users[i].password, users[i].businessInfo);

switch (strength) {

case 3: printf("Strong\n"); break;

case 2: printf("Medium\n"); break;

case 1: printf("Weak\n"); break;

default: printf("Very Weak\n"); break;

}

}

}

void simulatePhishingAttack() {

const char \*fakeEmails[MAX\_EMAILS] = {

"user1@cloudstrike.com",

"user2@cloudstrike.com",

"user3@cloudstrike.com",

"admin@cloudstrike.com",

"support@cloudstrike.com",

"info@cloudstrike.com",

"contact@cloudstrike.com",

"security@cloudstrike.com",

"help@cloudstrike.com",

"sales@cloudstrike.com"

};

printf("\nSimulated Phishing Emails:\n");

for (int i = 0; i < MAX\_EMAILS; i++) {

printf("Fake Email: %s, Effectiveness: %s\n",

fakeEmails[i], (i % 2 == 0) ? "High" : "Low");

}

}

void identifyNetworkVulnerabilities(NetworkDevice devices[], int deviceCount) {

printf("\nPotential Vulnerabilities in Network Security:\n");

for (int i = 0; i < deviceCount; i++) {

if (strstr(devices[i].vulnerability, "unpatched") != NULL) {

printf("Device '%s' has unpatched vulnerabilities.\n", devices[i].device);

}

if (strstr(devices[i].vulnerability, "weak firewall") != NULL) {

printf("Device '%s' has weak firewall rules.\n", devices[i].device);

}

if (strstr(devices[i].vulnerability, "insufficient segmentation") != NULL) {

printf("Device '%s' has insufficient network segmentation.\n", devices[i].device);

}

}

}

int main() {

FILE \*

FILE \*file = fopen("credentials.txt", "r");

if (!file) {

perror("Failed to open credentials file");

return EXIT\_FAILURE;

}

UserCredential users[MAX\_USERS];

int userCount = 0;

while (fgets(users[userCount].username, sizeof(users[userCount].username), file) != NULL) {

char \*token = strtok(users[userCount].username, ",");

if (token != NULL) {

strcpy(users[userCount].username, token);

token = strtok(NULL, ",");

if (token != NULL) {

strcpy(users[userCount].password, token);

token = strtok(NULL, "\n");

if (token != NULL) {

strcpy(users[userCount].businessInfo, token);

userCount++;

}

}

}

}

fclose(file);

evaluateCompromisedPasswords(users, userCount);

identifyPasswordVulnerabilities(users, userCount);

simulatePhishingAttack();

file = fopen("network\_devices.txt", "r");

if (!file) {

perror("Failed to open network devices file");

return EXIT\_FAILURE;

}

NetworkDevice devices[MAX\_DEVICES];

int deviceCount = 0;

while (fgets(devices[deviceCount].device, sizeof(devices[deviceCount].device), file) != NULL) {

char \*token = strtok(devices[deviceCount].device, ",");

if (token != NULL) {

strcpy(devices[deviceCount].device, token);

token = strtok(NULL, "\n");

if (token != NULL) {

strcpy(devices[deviceCount].vulnerability, token);

deviceCount++;

}

}

}

fclose(file);

identifyNetworkVulnerabilities(devices, deviceCount);

return EXIT\_SUCCESS;

}