Dokumentation Aufgabe 4: Bericht zu gefragten Punkten

1. Vollständiger Quellcode mit ausführlichen Kommentaren

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// berechne ein pseudo hash value basierend auf dem passwort
unsigned int rechneHash(const char *password) {
    unsigned int hash = 0;

    //automatisch int overflow provozieren, demozwecke
    if (strcmp(password, "IntegerOverflow") == 0) {
        //2^32 + 42 = ergibt durch overflow: 42
        unsigned long long grossWert = 4294967338ULL; //ULL unsigned long long

        //expliziter cast auf signed int. Nur die unteren 32 bit bleiben
        hash += (unsigned int)grossWert; // hier entsteht der overflow bewusst

        printf("\nInteger Overflow demonstration.\n");
        printf("addition von 4294967338 (2^32 + 42) zu unsigned int Hash.\n");
        printf("durch overflow wird nur der untere 32 bit teil verwendet:\n");
        printf("(unsigned int)(4294967338) = %u\n\n", hash);

        return hash;
    }

    //summiere ASCII werte der pw-zeichen
    int len = strlen(password);
    for (int i = 0; i < len; i++) {
        hash += (unsigned int)password[i];
    }

    return hash;
}

// auth funktion - vergleicht berechneten hash mit dem festgesetzten wert (42)
int auth(const char *password) {
    // berechne den hash des input pw
    unsigned int hash = rechneHash(password);

    // vergleiche mit dem erwarteten Hash-Wert
    // zum testzweck: korrektes pw ergibt einen hash von 42
    if (hash == 42) {
        printf("auth erfolgreich. Sup\nHash: %u\n", hash);
        return 1;
    } else {
        printf("auth fail. Hash: %u\n", hash);
        return 0;
    }
}

int main() {
    char password[256]; // buffer für passwordinput

    printf("HINWEIS: der Hash des Passworts muss 42 ergeben\n");
    printf("Für Demozwecke: gib 'IntegerOverflow' ein, um automatisch einen Overflow zu testen\n");
    printf("Passwort eingeben: ");

    if (fgets(password, sizeof(password), stdin) == NULL) {
        printf("error beim input.\n");
        return 1;
    }

    // remove (\n) aus der eingabe
    password[strcspn(password, "\n")] = '\0';

    // auth versuchen
    auth(password);

    return 0;
}
```

2. Verwendete Compiler-Flags und Begründung der Auswahl

gcc -o ioverflow ioverflow.c -Wall -Wextra -m32

-Wall -Wextra: Aktivieren basic und erweiterte Warnungen für Codeüberprüfung
-O: Keine Optimierung, wichtig für die saubere Demonstration vom Overflow

3. Schritt-für-Schritt-Anleitung zur Umgehung der Authentifizierung

Voraussetzung: auth() muss hash == 42 zurückgeben.

Standardweise wird der Hash aus ASCII Summen berechnet:

hash+=(unsigned int)password[i];

Normalerweise war es vorgesehen, eine Textdatei mit einem sehr großen Passwort einlesen zu lassen, um den Integer Overflow zu demonstrieren. Allerdings ermöglicht die Dateigrößenbegrenzung auf Moodle diesen Weg nicht. Daher wurde zu Demonstrationszwecken ein alternativer Weg im Code implementiert.

Die spezielle Eingabe “IntegerOverflow” führt zu folgendem Verhalten:

unsigned long long grossWert = 4294967338ULL; // 2^32 + 42
hash+=(unsigned int)grossWert; //ergibt durch Overflow: 42

Unsigned long long ist ein Datetyp für sehr große positive Ganzzahlen mit 64 bit.
Suffix ULL bei 4294967338ULL teilt dem Compiler mit, dass es sich um eine Konstante vom Typ *unsigned long long* handelt. Da ein *unsigned int* nur 32 bit groß ist, kann es maximal den Wert *4294967295* speichern. Wenn man einen größeren Wert (hier *4294967338*) in einen *unsigned int* umwandelt, wird nur der untere 32 bit Teil übernommen. Der Rest geht verloren, was zum **Integer Overflow** führt und hier den Wert *42* gibt. Das Passwort ist somit theoretisch falsch aber es erfolgt trotzdem eine erfolgreiche Authentifizierung. Das Ergebnis ist somit:

4294967338 mod 2^32 = 42

4. Mathematische Erklärung des Integer Overflow-Mechanismus

Overflow bei unsigned int(32 bit): der Wertebereich ist gültig von 0 bis 2^32 – 1, also von 0 – 4294967295. Wird ein Wert darüber hinaus in ein unsigned int gecasted, wird module 2^32 gerechnet:

4294967338 % 2^32 = 4294967338 – 2^32 = 42

Im Falle des Codes:

***grossWert**: 4294967338*
***2^32**: 4294967296*
*4294967338 – 4294967296 = **42***

5. Beispiele verschiedener Eingabewerte

Eingabe	ASCII Summe (also der Hash)	Authentifizierung erfolgreich?
*	42	Ja
42	102	Nein
abc	97+98+99 = 294	Nein
!!!!!!!	7*33 = 231	Nein
IntegerOverflow (4294967338ULL)	2^32 + 42 = 42	Ja

6. Präventationsmaßnahmen gegen Integer Overflow

1. Typwahl bewusst treffen: Verwenden von size_t für Indizes und unsigned Types nur wenn absolut notwendig

2. Einen möglichen Overflow überprüfen, indem man Grenzwerte prüft, also z.B:

if (UINT_MAX – hash < val)

3. Richtige Hash-Algorithmen benutzen, also statt einfacher ASCII-Summe lieber MD5 oder SHA256

7. Präventationsmaßnahmen gegen Integer Overflow

TYP	Wertebereich	Overflow behavior
int(signed)	-2147483648 bis 2147483647	undefiniert
Unsigned int	0 bis 4294967295	Definiert – mod 2^32
short	-32767 bis 32767	undefiniert
Long long	+-9223372036854775807	undefiniert

Generell gilt

Signed Typen: undefiniertes Verhalten
Unsigned Typen: definiertes Verhalten