Testing und Debugging: Übung 10

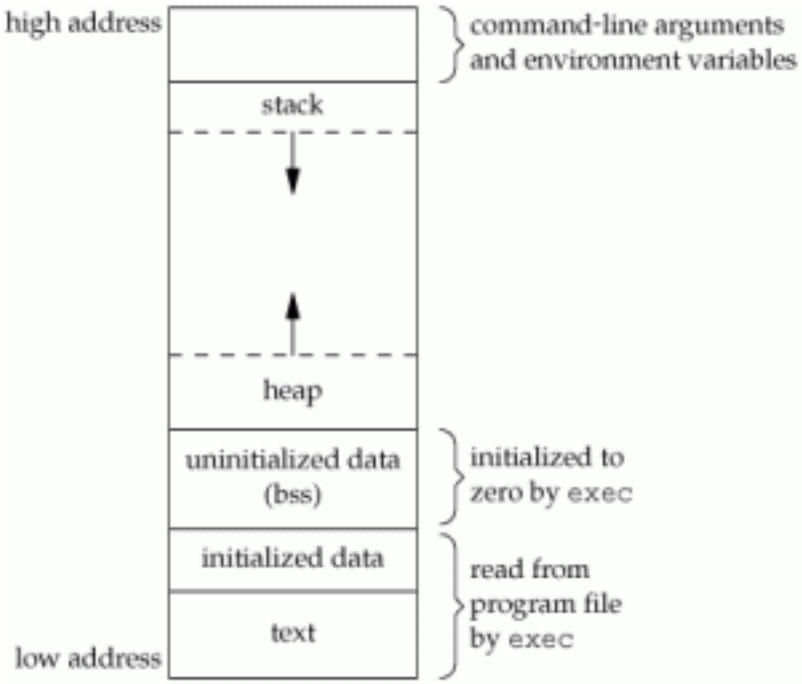
Abgabetermin siehe Canvas

Warnung
Aufgrund der Länge der Aufgaben sind 50 Punkte der schriftlichen Aufgaben Bonuspunkte.

1 Memory Probleme

Beim Programmieren in Sprachen wie *C* haben Sie die Möglichkeiten, den, dem Programm zugewiesenen (virtuellen) Arbeitsspeicher auf die verschiedensten Weisen zu beeinflussen.

Die schematische Darstellung des Speichers eines laufenden Programms:



(Quelle und genauere Beschreibung: http://www.geeksforgeeks.org/memory-layout-of-c-program/)

Gegeben seien die folgenden beiden Quelldateien, welche zusammen mit *uebung.c*, *middle.c* und *functions.h* die Basis der aktuellen Übung (schriftlicher Teil und Programmieren) bilden:

center.c

```c
#include <string.h>
/* Die Funktion zentriert den String s innerhalb einer 80 Zeichen Ausgabe */
char *center(char *s){
    int length;
    int i;
    char buf[81];   /* 81. Zeichen fuer \0 */

    length = strlen(s);

    /* Den Anfang des Strings mit Leerzeichen fuellen. */
    for (i = 0; i <= (78 - length) / 2; i++) {
        buf[i] = ' ';
```

```
    }

    /* Den String s an die Position nach den Leerzeichen kopieren. */
    strcpy(buf + i, s);

    return buf;
}
```

fib.c

```
int fib(int n) {
    int f;
    int f0 = 1;
    int f1 = 1;
    while (n > 1) {
        n = n - 1;
        f = f0 + f1;
        f0 = f1;
        f1 = f;
    }
    return f;
}
```

Achtung
Arbeiten Sie bitte ausschließlich in der virtuellen Maschine, da sich anderen falls nicht alle Fehler so zeigen wie es für die Aufgabe gedacht war.

Kompilieren Sie die *C* Dateien mit dem mitgeliefertem *Makefile*, bevor Sie mit der Bearbeitung der schriftlichen Aufgaben beginnen.

1.1 Bufferoverflow (50 Punkte)

Zu einem Bufferoverflow kommt es, wenn man ein Array über seine Grenzen hinaus mit Daten befüllt oder davon liest. Dies kann unvorhersagbare Zustände im Programm auslösen. Unter Umständen kann dadurch auch versehentlich (oder mit Absicht) Code ausgeführt werden, der gar nicht dazu gedacht war an dieser Stelle ausgeführt zu werden.

Im beiliegendem *perl* Skript wird ein solcher Bufferoverflow in der *center* Funktion ausgenutzt, um die *usage* Funktion auszuführen.

Lesen Sie sich den Kommentar in der Datei *bufferoverflowexploit* durch, passen das Skript entsprechend an und führen es wie dort beschrieben aus.

Aufgaben

1. *Welches Fehlverhalten können Sie an der Ausgabe der* center *Funktion in der Datei* ergebnis.txt *neben dem Auslösen der* usage *Funktion noch beobachten? (30 Punkte)*

```
Erwartet:
.
.
.
Beobachtet:
.
```

```
.
.
.
Fehlerursache:
.
.
.
.
.
.
```

2. *Welche Zeile in der* center.c *Datei verursacht den Bufferoverflow? Begründen Sie Ihre Antwort. Durch welche Änderung in dieser Zeile kann man den Bufferoverlow verhindern? (10 + 10 Punkte)*

```
.
.
.
.
.
.
.
.
.
.
.
.
.
```

1.2 Lokalisieren der Variablen im Speicher (100 Punkte)

Je nachdem in welchem Bereich des Speichers (siehe obige Grafik) sich eine Variable befindet, kann sie für verschiedene Fehlertypen (mit-) verantwortlich sein.

Tipp
Für die 3. Frage kann die Benutzung des GNU-Debuggers hilfreich sein. Die Datei `GDB_Cheat_Sheet.pdf` enthält eine Kurzanweisung für den GNU-Debugger.
Den Debugger starte man (nach dem Kompilieren mit *-g* Flag; in Makefile schon gesetzt) wie folgt:

```
gdb ./uebung10
```

Man erhält dann einige Informationen über den Debugger und dann den Prompt:

```
(gdb)
```

Dann setzen Sie Breakpoints in den Zeilen 77, 78, 80. Breakpoints sorgen während der Ausführung im Debugger dafür, dass das Programm **vor** dem Ausführen der angegebene Zeile anhält.

```
b 77
b 78
b 80
```

Das Setzen wird jeweils mit einer Antwort der Form (die Hexadezimalzahl kann bei Ihnen abweichen)

```
breakpoint 1 at 0x400767: file uebung10.c, line 77.
```

Quittiert. Anschließend starten Sie das Programm mit den Parametern „1 2 3 hallo“:

```
r 1 2 3 hallo
```

Das Programm wird bis zum 1. Breakpoint ausgeführt und hält nun an. Zum Beobachten der Variablen geben Sie folgende Befehle ein:

```
display msg
display *msg@81
```

Der erste Befehl zeigt die Speicheradresse und den Inhalt der Variable (bis zum String-ende Symbol *\0*) an. Der Zweite Befehl ist praktisch, wenn man sich eine bestimmt Anzahl von Einträgen in einem Array anzeigen lassen will. Dabei dereferenziert der Stern den Pointer *msg* und *@81* gibt die Anzahl der anzuzeigenden Felder an. Der *display* Befehl hat den Vorteil gegenüber dem *print* Befehl, dass nach jeder ausgeführten Zeile im Programm, welche die angegebenen Variablen verändert, *display* die aktualisierten Daten erneut ausgibt; *print* tut dies nicht.
Mit *c* kann man nun das Programm bis zum nächsten Breakpoint fortsetzten. Mit *n* kann man im Einzelschrittmodus durch das Programm gehen, wobei der debugger nicht in eine aufgerufene Funktion selbst einsteigt. Wenn man auch in jede Funktion mit absteigen möchte, so sollte man *s* statt *c* verwenden.
Zum Beenden des Debuggers benutzt man das Kommando *q*.

Aufgaben

1. *In welchem Teil des Speichers sind die Variablen* length*,* i *und* buf *der Funktion* center *angesiedelt, wenn diese Funktion ausgeführt wird? (10 Punkt)*

```
.
.
```

2. *In welchem Zeitraum während der Ausführung des Programms existieren die o. g. Variablen? (10 Punkte)*

.
.

3. *Wohin zeigt die Variable msg nachdem die Zeile 77 von uebung10.c ausgeführt wurde? Worauf nach Zeile 78 von uebung10.c? Was steht jeweils an diesen Speicherstellen? (5 + 5 + 5 Punkte)*

.
.
.
.
.
.
.
.
.
.
.
.

4. *Welche Werte kann* i *annehmen, bevor die Variable in der Zeile mit* strcpy *benutzt wird ? (10 Punkte)*

.
.
.

5. *In welchem Teil des Speichers sind die Variablen* f, f0 *und* f1 *der Funktion* fib *angesiedelt, wenn diese Funktion ausgeführt wird? (10 Punkt)*

.
.

6. *In welchem Zeitraum während der Ausführung des Programms existieren die o. g. Variablen? (10 Punkte)*

.
.

7. *Auf welche Adresse zeigt die Variable* f *aus* `uebung10.c` *vor und nach der Ausführung der* fib *Funktion? Auf welche Adresse zeigt die Variable* f *in der* fib *Funktion vor dem return? (in gdb jeweils "*p &f*" an geeigneter Stelle ohne Anführungszeichen ausführen) (5 + 5 + 5 Punkte)*

.
.
.
.
.
.
.
.
.
.
.
.

8. *Welche Werte kann* f *in* fib *annehmen, bevor sie mit* return *zurückgegeben wird? (10 Punkte)*

```
.
.
.
```

9. *Ruft man das Programm mit negativen Zahlen für die ersten drei Parameter auf, so gibt* fib *unerwartete Werte zurück. Warum ist das so? Wo liegt der Fehler in* fib.c*? (10 Punkte)*

```
.
.
.
.
.
.
.
```

2 Programmieraufgabe: Beheben der drei Fehler im o.g. Programm. (100 Punkte)

Konzeptionelle Beschreibung der Aufgabe

Dateien für die Programmieraufgabe

Grading/*init*py	Hinweisdatei für Python: Das Verzeichnis ist ein Modul
Grading/Grading.py	Quellcode der Testklasse, die in *run_test.py* verwendet wird
uebung10.c *	In dieser Datei müssen Sie nur eine kleine Änderung für den Fix von *center* einfügen und Ihre Daten eintragen
center.c *	In dieser Datei müssen Sie zwei Fehler fixen
fib.c *	In dieser Datei müssen Sie einen Fehler fixen
middle.c	In dieser Datei ist die *middle* Funktion aus der letzten Übung in *C* korrekt implementiert
functions.h	Header für die drei Funktionen.

Die mit [*] markierten Dateien müssen **bearbeitet** und **abgegeben** werden.

2.1 Aufgabe

- In der *center* Funktion ist der Bufferoverflow zu beheben.
- Um die ungewollten Sonderzeichen in der Ausgabe der *center* Funktion zu verhindern soll der Speicherbereich der Variable *buf* nicht auf dem Stack, sondern auf dem Heap liegen. Bei dieser Änderung müssen Sie darauf achten, das **vor dem erste** und **nach dem letzten** benutzen des Inhalts von *buf* bestimmte Massnahmen getroffen werden müssen, die sicherstellen, dass der Speicherbereich benutzbar ist bzw. das der Speicherbereich wieder aufgeräumt wird.
- Die Funktion *fib* hat ein unvorhersagbares Verhalten wenn Argumente < 2 verwendet werden. Sorgen Sie dafür, das sich die Fibonacci Folge auch für diese Eingaben gemäß ihrer definition verhält.

2.2 Bewertung

Je 25 Punkte für die Behebung des Bufferoverflows und für das Korrigieren der Funktion *fib*. 50 Punkte für die Behebung des zweiten Fehlers in *center* nebst den angedeuteten, zusätzlichen Maßnahmen.

Viel Erfolg