

☐ Gr. 1, Dr. PitzerName Thomas RingerAufwand in h 5☒ Gr. 2, Winkler, BSc Msc

Punkte _____ Kurzzeichen Tutor / Übungsleiter _____ / _____

Beispiel	L Lösungsidee	I Implementierung	T Tests	S = L+I+T	M Multiplikator	S*M
1	0..6	0..3	0..1	1	1	1
2 a	0..5	0	0..5	1	2	2
2 b	0..2	0..5	0..3	1	2	2
3	0..3	0..4	0..3	1	3	3
4	0..3	0..4	0..3	1	2	2
Summe (Erfüllungsgrad) Bitte online ausfüllen!						10

Bitte erstellen Sie für diese Hausübung ein ZIP-Archiv in dem im Unterverzeichnis „doc“ die gesamte Dokumentation (inkl. Lösungsidee und formatiertem Quelltext) als PDF vorliegt, sowie die Quelltexte in verschiedenen Unterverzeichnissen unterhalb des Verzeichnis „src“.

1. Typkonvertierungen (src/type-conversion/)

Überlegen Sie bei den folgenden Beispielen welche impliziten Typkonvertierungen erfolgen würden und schreiben Sie (falls möglich) dann ein geeignetes Testprogramm das Ihre Annahmen überprüft.

```
//a)
int i = 12; float f = 12.25;
//Welche Konvertierungen werden durchgeführt und wie lautet das Ergebnis?
double d = i + f;
```

```
//b)
int i = 1; char c = 'a';
//Welche Konvertierungen werden durchgeführt und wie lautet das Ergebnis?
int j = c + i;
```

```
//c)
int i = 10; int j = 3; float f = 10.25;
//Welche Konvertierungen werden durchgeführt und wie lautet das Ergebnis?
//Was sollte man hier verbessern?
double d = f + i / j;
```

```
//d)
char c; int i; float f; double d;
//Welchen Typ hat dieser Ausdruck?
(c + i) * (f / d);
```

```
//e)
int x = -25;
unsigned int y = 10;
//Was wird hier ausgegeben? Warum?
if( (x+y) < 0) printf("A\n");
else printf("B\n");
```

2. Buffer-Overflow: Hack and Fix (src/buffer-overflow/)

- a) Die ausführbare Datei `buffer-overflow` im ZIP-Archiv „`buffer-overflow-executable.zip`“ hat eine gravierende Sicherheitslücke. Das Programm gibt normalerweise nur bei Eingabe des richtigen Benutzernamens und Passworts einen Code aus, mit dem Sie das ZIP-Archiv „`buffer-overflow-sources.zip`“ entschlüsseln können. Darin befindet sich dann der dazugehörige Quelltext der beiden Programme. Starten Sie das Programm und versuchen Sie durch Ausnutzen eines Buffer-Overflows auch ohne die Angabe der richtigen Zugangsdaten trotzdem den Code für das ZIP-Archiv mit dem Quelltext zu bekommen. Dokumentieren Sie Ihr Vorgehen.

Hinweis: Falls Sie keine Möglichkeit finden das Programm durch einen Pufferüberlauf auszutricksen können Sie alternativ versuchen das richtige Passwort und auch den Code für das ZIP-Archiv durch Auslesen aller eingebetteter Zeichenketten aus dem Executable mit dem UNIX-Kommando „`strings`“ zu erhalten.

- b) Sobald Sie dann den Quelltext erhalten haben, ersetzen Sie die Verwendung möglichst vieler unsicherer Funktionen, wie `strcpy()`, durch ihre sichereren Varianten. Bei der Verwendung von `scanf()` kann durch eine entsprechende Formatangabe ebenfalls verhindert werden, dass über den angegebenen Puffer hinaus geschrieben wird. Testen Sie das modifizierte Programm ob die ausgenutzte Sicherheitslücke aus Aufgabe (a) behoben ist.

3. Minimum und Maximum ermitteln (src/minmax/)

Schreiben Sie ein C-Programm `MinMax`, das Folgendes leistet: Dem Programm *MinMax* sollen in der Kommandozeile beliebig viele ganze Zahlen mitgegeben werden können. Das Programm soll die kleinste negative (= Minimum) sowie die größte positive (= Maximum) der als Parameter übergebenen Zahlen als Ergebnis ausgegeben. Kommen in der Parameterliste keine negativen Zahlen vor, soll für das Minimum Null („0“) ausgegeben werden. Analog für das Maximum, falls keine positiven Zahlen in der Parameterliste vorkommen.

Beispiele:

Aufruf: `minmax -2 17 -4 -5`

Ausgabe: `minimum = -5 maximum = 17`

Eingabe: `minmax`

Ausgabe: `minimum = 0 maximum = 0`

4. Fehler verschluckt? (src/errno/)

Wenn Sie die Handbuchseite (manpage) von `atoi` studieren, fällt sofort auf, dass diese Funktion keinerlei Fehlerbehandlung bietet. Versuchen Sie anhand der Dokumentation eine Alternative zu finden und erweitern Sie das Programm `minimax`, sodass eine fehlerhafte Eingabe erkannt wird und eine entsprechende Meldung erscheint. Finden Sie dazu heraus, was es mit `ERRNO` auf sich hat, und beschreiben Sie, wie dieser Mechanismus funktioniert.

Hinweise:

1. Lesen Sie die organisatorischen Hinweise.
2. Geben Sie für alle Ihre Lösungen immer eine Lösungsidee an.
3. Kommentieren und testen Sie Ihre Programme ausführlich.

Lösungsideen

Contents

1	Typkonvertierungen	1
1.1	a	1
1.2	b	1
1.3	c	2
1.4	d	2
1.5	e	2
2	Buffer-Overflow: Hack and Fix	2
3	Minimum und Maximum ermitteln	3
4	Fehler verschluckt.	3
5	Code	4
5.1	Typeconversion	4
5.2	Buffer Overflow hack	4
5.3	MinMax	5
5.4	MinMax mit Errors	5
6	Tests	6
6.1	TypeConversions	6
6.2	Buffer Overflow Hack and Fix	6
6.3	MinMax	6
6.4	MinMax mit Fehler	6

1 Typkonvertierungen

1.1 a

Die Konvertierung wird hier von integer auf float durchgeführt und das ergebnis dann auf double convertiert. Das Ergebnis ist 24.25

1.2 b

Hierbei findet die Konvertierung von char auf int statt und das Ergebnis lautet 97 da 'a' die Zahl 96 im Ansii Table representiert

1.3 c

hierbei wird zuerst eine int division durchgeführt bei der das Ergebnis 3 lautet (sprich die kommas-tellen werden einfach weggeschnitten) und danach bei der addition der int in float umgewandelt was ein ergebnis von 13.25 liefert und das ergebnis als double gespeichert. Eine Verbesserung hierbei wäre einen der beiden int explizit auf float zu casten: `double j = f + (float)i / j`; Dies liefert das gewünschte/richtige Ergebnis 13.5833333.

1.4 d

Float wird zu double Konvertiert, char zu int und danach int zu double bei der Multiplikation.

1.5 e

Hier ist das ergebnis der 'else' Zweig (also 'B'), da bei der kombination von signed und unsinged int der signed zu unsinged "konvertiert" wird und -25 ein einfach 25 vom INT_MAX abzieht.

2 Buffer-Overflow: Hack and Fix

Die idee hierbei war zu anfang einmal probieren ob der Code theoretisch richtig funktionieren könnte, wenn man sich an die schriftlichen limitationen hält. Habe somit weniger als 10 zeichen eingegeben und wie Erwartet die negative Antwort des Programms erhalten.

```
thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./a.out admin
password (max 10 chars): dsfdsaf
username = "admin", password = "dsfdsaf", access token = 0 (0x0, '')
You need access token = 97 (= 0x61 [hex], 'a' [ascii]) to access the ZIP-File password
```

Fig. 1: Erwarteter Error.

Die nächste Vorgehensweise war eben den Buffer Overflow zu erzeugen indem man eine Zeichenkette eingibt die das vordefinierte Array des Programmes überschreitet, und somit da Passwort freigibt.

```
thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./buffer-overflow admin
password (max 10 chars): iunliujnozshbdfkuasdzbfkjsuabvfjshgbfjuszaghfsa
username = "hdbfkwasdzbfkjsuabvfjshgbfjuszaghfsa", password = "iunliujnozshbdfkuasdzbfkjsuabvfjshgbfjuszaghfsa", access
token = 102 (0x66, 'f')
You need access token = 97 (= 0x61 [hex], 'a' [ascii]) to access the ZIP-File password
Segmentation fault (core dumped)
```

Fig. 2: Buffer-Overflow in action

Dies hat bei meinem ersten Versuch perfekt funktioniert und ich konnte das Passwort auslesen. Allerdings jetzt beim Screenshotten für die Doku funktioniert es aus unerfindlichen Gründen nicht mehr. Somit werde ich zusätzlich hier die Zweite möglichkeit testen: strings. Dies gibt mir die strings der Kompilierten Datei aus und dabei auch das Passwort.

```

thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ strings buffer-overflow
/lib64/ld-linux-x86-64.so.2
libc.so.6
strcpy
__isoc99_scanf
puts
printf
__cxa_finalize
strcmp
__libc_start_main
GLIBC_2.7
GLIBC_2.2.5
__ITM_deregisterTMCloneTable
__gmon_start__
__ITM_registerTMCloneTable
u/UH
[]A\A]A^A
usage: buffer-overflow <username>
e.g. "buffer-overflow admin"
    then enter your password when prompted
password (max 10 chars):
admin
letmein
username = "%s", password = "%s", access token = %d (0x%x, '%c')
FULL ACCESS GRANTED
Source Code Password is "BuFFeR0vErFl0w"
You need access token = %d (- 0x%x [hex], '%c' [ascii]) to access the ZIP-File password
; *3$
GCC: (Ubuntu 8.3.0-6ubuntu1) 8.3.0
crtstuff.c

```

Fig. 3: Passwortauslese mit strings

Damit hat es funktioniert und das Programm kann verbessert werden. Was im Endeffekt bedeutet sämtliche Vorkommnisse von `strcmp` zu `strncmp` umzuschreiben. Der Versuch danach wieder einen BufferOverflow zu erzwingen ist erwarteterweise gescheitert:

```

thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./a.out admin
password (max 10 chars): seurfjlsgrdnmflks.mdöfcokvlsawerf^[[3~
username = "admin", password = "seurfjl", access token = 0 (0x0, '')
You need access token = 97 (= 0x61 [hex], 'a' [ascii]) to access the ZIP-File password

```

Fig. 4: Buffer Overflow Fixed

3 Minimum und Maximum ermitteln

Hierbei ist die Idee ein Programm zu schreiben dem man so viele Argumente mitgeben kann wie man will, und danach diese Elemente mithilfe der For Schleife miteinander vergleicht und am Ende ein Min und ein Max ausgibt. Hierbei werden die Argumente die zum Programmstart mitgegeben werden von strings (bzw. char arrays) mittels `atoi` in `int` konvertiert um diese auch vergleichen zu können. Für den Sourcecode hierzu siehe Abschnitt 5.3.

4 Fehler verschluckt.

Anhand der Dokumentation habe ich ermittelt, genau wie die Angabe vermuten lässt, `atoi` keinerlei Fehlerbehandlung enthält. Eine Alternative dazu war `strtol` (string to long) welches den

string anstelle eines int in einen long int umwandelt und zusätzlich fehlerbehandlung über errno enthält.

Errno selbst wird über ein Header file eingefügt und ist im endeffekt nur ein int der vor der Verwendung von strtol mit 0 (null) initialisiert wird und der wert danach nicht mehr 0 ist.

Mit diesem Wissen wurde dann das MinMax Programm vom Vorpunkt umgebaut um ein mindestmaß an Fehlerbehandlung zu beinhalten. Siehe hierzu Abschnitt 5.4.

5 Code

5.1 Typeconversion

hierzu kein code vorhanden (für Validierungen der Annahmen a bis e ohne d siehe Abschnitt 6.1)

5.2 Buffer Overflow hack

```

1  #include <stdio.h>
2  #include <string.h>
3  // NOTE: needs to be compiled with GCC option -fno-stack-protector to be
   exploitable
4  int main(int argc, char *argv[]) {
5      char access_token = 0;
6      char name[11];
7      char password[11];
8
9      if (argc != 2) {
10         printf("usage: buffer-overflow <username>\n");
11         printf("e.g. \"buffer-overflow admin\\\"\\n");
12         printf("      then enter your password when prompted\\n");
13         return -1;
14     }
15     strncpy(name, argv[1], 5);
16     printf("password (max 10 chars): ");
17     scanf("%7s", password);
18
19     if (strncmp(name, "admin", 5) == 0 &&
20         strncmp(password, "letmein", 7) == 0) {
21         access_token = 97;
22     }
23
24     printf("username = \"%s\\", password = \"%s\\", access token = %d (0x%x, '%c')\\n", name, password, access_token, access_token, access_token);
25     if (access_token == 97) {
26         printf("FULL ACCESS GRANTED\\nSource Code Password is \"BuFfeR0veRFloW\\\"\\n");
27     } else {
28         printf("You need access token = %d (= 0x%x [hex], '%c' [ascii]) to access the ZIP-File password\\n", 97, 97, 97);
29     }
30     return 0;

```

```
31 }
```

5.3 MinMax

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main (int argc, char* argv[]) {
5     int min = 0, max = 0, number_to_compare;
6     for (int i = 1; i < argc; i++)
7     {
8         number_to_compare = atoi(argv[i]);
9         max = number_to_compare > max ? number_to_compare : max;
10        min = number_to_compare < min ? number_to_compare : min;
11    }
12    printf("Minimum: %d, Maximum: %d \n", min, max);
13    return 0;
14 }
```

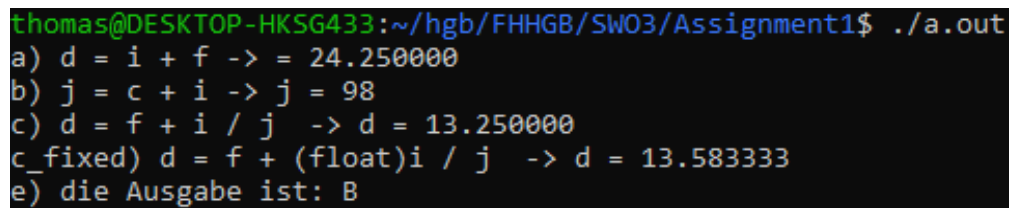
5.4 MinMax mit Errors

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <errno.h>
4 #include <limits.h>
5 int main (int argc, char* argv[]) {
6
7     long int min = 0, max = 0, number_to_compare;
8     for (int i = 1; i < argc; i++)
9     {
10        errno = 0;
11        char * end;
12        number_to_compare = strtol(argv[i], &end, 10);
13
14        if ((errno == ERANGE
15            && (number_to_compare == LONG_MAX || number_to_compare ==
16                LONG_MIN))
17            || (errno != 0 && number_to_compare == 0))
18        {
19            perror("strtol");
20            exit(EXIT_FAILURE);
21        }
22        if (argv[i] == end) {
23            printf("No Valid numbers found\n");
24            exit(EXIT_FAILURE);
25        }
26        max = number_to_compare > max ? number_to_compare : max;
```

```
27     min = number_to_compare < min ? number_to_compare : min;
28 }
29 printf("Minimum: %ld, Maximum: %ld \n", min, max);
30 return 0;
31 }
```

6 Tests

6.1 TypeConversions



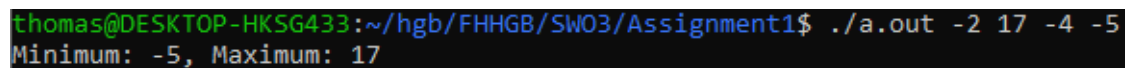
```
thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./a.out
a) d = i + f -> = 24.250000
b) j = c + i -> j = 98
c) d = f + i / j -> d = 13.250000
c_fixed) d = f + (float)i / j -> d = 13.583333
e) die Ausgabe ist: B
```

Fig. 5: Konvertierungs Ergebnisse

6.2 Buffer Overflow Hack and Fix

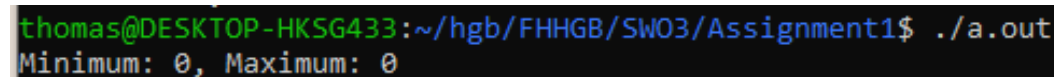
Siehe Figures 1, 2, 3 und 4.

6.3 MinMax



```
thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./a.out -2 17 -4 -5
Minimum: -5, Maximum: 17
```

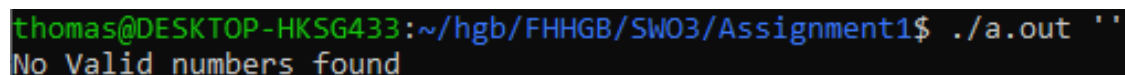
Fig. 6: MinMax Test1



```
thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./a.out
Minimum: 0, Maximum: 0
```

Fig. 7: MinMax Test2

6.4 MinMax mit Fehler



```
thomas@DESKTOP-HKSG433:~/hgb/FHHGB/SW03/Assignment1$ ./a.out ''
No Valid numbers found
```

Fig. 8: MinMax mit Fehlerausgabe