TI III WS 2013, Fr. 12-14

Lösung Übungsblatt 8

Christoph van Heteren-Frese (Matr.-Nr.: 4465677), Julien Stengel
Tutor: Ruhland, eingereicht am 13. Dezember 2013

Aufgabe 3

Als Bufferoverlow wird eine Sicherheitslücke bezeichnet, bei der zu große Datenmengen in einen dafür zu kleinen reservierten Speicherbereich geschrieben werden. Als Konsequenz werden die hinter dem Ziel-Speicherbereich liegenden Speicherstellen überschrieben. Gelingt es auf diese Weise Rücksprungadressen von Funktionen zu überschreiben, kann somit fremder Code eingeschleust und ausgeführt werden.

Beispiel:

```c
/* Compiliert mit gcc (Ubuntu/Linaro 4.8.1-10ubuntu9) 4.8.1*/

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

int main() {
    char buffer[1];

    /* Grundaetzlich kein Zugang gestattet*/
    int access = 0;

    printf("Input: ");
    scanf("%s", &buffer);

    /* Ueberpruefung, ob user zugangsberechtigt...*/
    if (access != 0)
        printf("Access granted!");
    else
        printf("Access denied");

    printf("\nInhalt von access: 0x%x (hex, little endian)\n",access);
    return 0;
}
```

Listing 1: overflow.c

Das Beispiel in Listing 1 verdeutlicht die Gefahr, die von einem Bufferoverflow ausgehen kann. Es wird eine Eingabe unbestimmter Länge gelesen. Ist die Eingabe länger als ein Zeichen, wird die Variable `access` überschrieben . Dies führt zur Verfälschung einer Sicherheitsüberprüfung.

Aufgabe 4

a)

```
PRINT_INTEGER(square(1+1));
```

in Zeile 17 von Listing 2 wird durch den Präprozessor durch

```
printf("%d ",1+1*1+1);;
```

ersetzt. Da `*` stärker bindet als `+` wird der Ausdruck `1+1*1+1` zu `3` ausgewertet.

b)

```
#define isdigit(c) (c>='0' && c<='9') ? 1 : 0
```

Siehe auch Listing 2 Zeile 7.

c)

```
printf("\nMaximum von a und b: %d\n",max(a,b++));
```

in Zeile 22 von Listing 2 wird durch den Präprozessor durch

```
printf("\nMaximum von a und b: %d \n",(a>b++) ? a : b++);
```

ersetzt. Der Ausdruck `(a>b++) ? a : b++` inkrementiert den Wert von `b` zwei Mal: einmal direkt nach dem Vergleich (dieser Wert wird direkt ausgegeben) und das zweite Mal nach der Ausgabe. Somit ist der Wert bei der zweiten Ausgabe zwei Mal erhöht.

d)

Vorteile von Makros liegen in der besseren Performance, die sich in einer Leistungssteigerung und einer kürzeren Laufzeit bemerkbar macht: Wird z.B. die Hälfte des Quelltextes bereits durch den Prozessor bearbeitet, ist nur noch die andere Hälfte vom Compiler zu bearbeiten. Ein weiterer Vorteil ist die ermöglichte Verbesserung der Lesbarkeit.

Nachteile: Der Leisterungssteigerung von Makros steht der erhöhte Speicherverbrauch entgegen. Des weiteren erhöht sich die Anzahl der möglichen Fehlerquellen.

e)

Es wird `VERBOSE nicht definiert` ausgegeben (an Stelle von `Test zu Ende!`, wenn die Zeile `#define VERBOSE` nicht entfernt wird).
Begründung: Mit der Präprozessoranweisung

```
#ifdef VERBOSE
```

in Zeile 26 wird abgefragt, ob das Symbol `VERBOSE` definiert wurde oder nicht. Ist das der Fall, wird der Block im Anschluß ausgeführt. Wenn das Symbol nicht definiert ist, wird der Block nach `#else` (Zeile 28) ausgeführt, so dass die oben genannte Ausgabe entsteht.

```
1  #include <stdio.h>
2
3  #define square(x) x*x
4  #define PRINT_INTEGER(i) printf("%d ",i);
5  #define max(a,b) (a>b) ? a : b
6  #define VERBOSE
7  #define isdigit(c) (c>='0' && c<='9') ? 1 :0
8
9  void main() {
10     int a,b,c;
11
12     #ifdef VERBOSE
13     printf("Makrotest\n");
14     #endif
15
16     PRINT_INTEGER(square(2));
17     PRINT_INTEGER(square(1+1));
18
19     a=1;
20     b=5;
21     /* b gleichzeitig um 1 erhöhen */
22     printf("\nMaximum von a und b: %d\n",max(a,b++));
23     printf("Wert von b: %d\n",b);
24     printf("isdigit: %d\n",isdigit('7'));
25
26     #ifdef VERBOSE
27     printf("Test zu Ende!\n");
28     #else
29     printf("VERBOSE nicht definiert\n");
30     #endif
31 }
```

Listing 2: u8_4.c

1.a)

Ja, die Freispeicherliste kann wiederhergestellt werden. Es gibt zwei Möglichkeiten. Die erste: der Table of Content der Platte wird durchgegangen und die freien Speicherplätze werden in einer Neuen Liste zusammen gefügt. Die zweite: die einzelnen Sektoren der Platte werden erneut darauf überprüft, ob sie beschrieben sind. Die leeren Sektoren werden in einer neuen Freispeicherliste zusammengefügt.

b)

Ein mögliches Verfahren wäre die Nutzung des RAID Level 1. Bei diesem Verfahren wird eine zweite Festplatte als exakte Kopie der ersten Platte genutzt, für den Fall, dass zu einem Speicherfehler des Zeigerverlustes (wie in a) beschrieben) kommen sollte , wird auf die zweite Platte zugegriffen, um die Verlorenen Daten wiederherstellen zu können. Außerdem werden noch andere Speicherfehler mit abgedeckt. Sollte keine zweite Platte vorhanden sein ließe sich durch ein Aufteilen der Platte in zwei gleichgroße Partitionen eine Art des RAID Level 1 auf einer Platte implementieren.

2.a)

FAT (file allocation table) ist eine Struktur, die es ermöglicht einer Datei zugeordnetem Speicher zurückverfolgen zu können. Dabei werden Dateien den entsprechenden Sektoren auf der Festplatte zugeordnet. FAT wird dabei in die Grundsätzlichen Teile des Bootsektors, Reservierte Sektoren, File Allocation Table, Wurzelverzeichnis und Datei- und Verzeichnis-Datenregion unterteil. FAT16 hat einen 16 Bit breiten Eintrag für die Sektorenanzahl und daher können maximal 512MB große Partitionen mit diesem Dateisystem verwaltet werden.

b)

Das Wurzelverzeichnis muss an einer bestimmten Stelle liegen, damit der Datenträger richtig gelesen werden kann. Daher muss das Wurzelverzeichnis statisch gewählt werden, damit das Betriebssystem den Datenträger richtig erkennen und nutzen kann. Die Unterverzeichnisse sind nicht essentiell für die Nutzung des Datenträgers und können daher beliebig auf dem Datenträger in beliebiger Größe angeordnet werden.

Da FAT16 für jede Datei 16 Bit für die Datei-Adressierung zur Verfügung hat, gibt es insgesamt 2^{16} mögliche Einträge => 65536 Dateien können in einem Unterverzeichnis angelegt werden.