Vorlesung Systemnahe Programmierung – WS 2014/15

Prof. Dr. Matthew Smith, Dr. Matthias Frank, Sergej Dechand

10. Übungszettel

Ausgabe: Mi 10.12.2014; Abgabe **bis zum (Freitag) 09.01.2015, 23:59 Uhr**
Die Vorführung erfolgt in der Woche **vom 12.01.2015 bis zum 16.01.2015** in Ihrer Übungsgruppe.

HINWEIS: Die vorlesungsfreie Zeit über die Weihnachts-/Neujahrspause geht von Mittwoch 24.12.2014 bis einschl. Dienstag 06.01.2015. Am Montag 22.12., Dienstag 23.12.2014 sowie Mittwoch 07.01. und Donnerstag 08.01.2015 finden ***KEINE* Übungsgruppen** zur Systemnahen Programmierung statt! Am Dienstag-Nachmittag **23.12.2014 findet *KEINE* Vorlesung** Systemnahe Programmierung statt.
Für dieses Übungsblatt haben Sie also (exklusive der Weihnachts-/Neujahrspause) mehr Bearbeitungszeit als sonst. Gegenüber der zulassungsrelevanten Bepunktung enthält dieser Übungszettel mehrere Bonus-Aufgaben.

Alle Programme müssen im Poolraum unter Linux kompilierbar, lauffähig und ausreichend kommentiert sowie mit einem Makefile versehen sein. Die Lösungen sind bei Ihrem Tutor während Ihrer Übungsgruppen vorzuführen. Die Abgabe erfolgt mittels Ihres SVN-Repositorys jeweils in dem Verzeichnis /tag/AufgabeXY.

Aufgabe 31: Sauberes Beenden von Serveranwendungen und Signalbehandlung (1+2+1 Punkte) BONUS-Aufgabe!

Hinweis: Dies ist eine Bonus-Aufgabe, d.h. die erreichten max. 4 Punkte erhöhen den Punktestand, aber das 50%-Zulassungskriterium wird nur auf die Summe aller Aufgaben OHNE Bonus-Aufgaben angewandt.

Programme von POSIX kompatiblen Betriebssystemen kommunizieren unter anderem über Signale (*Signals*). Diese Signale können genutzt werden, um Programme über die Ankunft neuer Daten, Ausnahmezustände des Betriebssystems oder Fehler im Programm zu informieren. Weiterhin kann ein Programm einem anderen per Signal mitteilen, es möge sich beenden.

a) Für Aufgabe 20 haben Sie einen iterativen TCP-Server entwickelt.
    a. Was geschieht, wenn während des Programmlaufs die Tastenkombination „Ctrl-C“ gedrückt wird?
    b. Welches ist die zu letzt ausgeführte Stelle in Ihrem Programmcode?
    c. Was geschieht, wenn während des Programmlaufs und aktiver Verbindung eines Clients die Tastenkombination „Ctrl-C“ gedrückt wird?
    d. Unter welchen Bedingungen können Sie den Server-Port in den Fällen a) und c) wiederverwenden?

b) Implementieren Sie einen sogenannten Signal-Handler (s. Folien 97-98 von Kapitel 1), der die Standardreaktion auf „Ctrl-C“ (*SIGINT*) abfängt und beenden Sie Ihr Programm sauber. Dazu zählt insbesondere das Schließen aller genutzten Sockets.

Tipp*:* Nutzen Sie dafür die Funktion *signal()*; s. Man-Page zu *signal()*.

c) Betrachten Sie erneut die Fragen aus Aufgabenteil a).

   a. Was hat sich geändert?

   b. Welche Erklärung haben Sie dafür?

Die Beantwortung der Fragen soll in einer README.txt Datei erfolgen.

Aufgabe 32: Matrixmultiplikation (4 Punkte)

Betrachten Sie das folgende C-Programm, welches Matrizen multipliziert:

```c
#include <stdio.h>


void print_matrix(int n, int m, int *M);

void mmul(int n, int m, int k, int *M1,
int *M2, int *M);


int main()
{
  int A []= {     /* 3x4-Matrix */
    1,2,-1,1,
    0,1,2,-2,
    2,-1,0,-3
  };

  int B [] = {    /* 4x2-Matrix */
    1,-2,
    0,-1,
    1,1,
    1,0
  };

  int C[6];       /* 3x2-Matrix */

  print_matrix(3,4,A);
  print_matrix(4,2,B);

  mmul(3,4,2,A,B,C);

  print_matrix(3,2,C);

  return 0;
}
```

```c
void print_matrix(int n, int m, int *M)
{
  int x;
  int y;
  printf("\n");
  for (y=0;y<n;y++) {
    for (x=0;x<m;x++) {
      printf ("%5d", M[y*m+x]);
    }
    printf("\n");
  }
  printf("\n");
}


void mmul(int n, int m, int k, int *M1,
          int *M2, int *M)
{
  int x;
  int y;
  int z;
  int temp;
  for (y=0;y<n;y++) {
    for (x=0;x<k;x++) {
      temp = 0;
      for (z=0;z<m;z++) {
        temp += M1[y*m+z]*M2[z*k+x];
      }
      M[y*k+x] = temp;
    }
  }
```

Das Programm finden Sie auch auf der Veranstaltungs-Homepage zum Download. Entfernen Sie aus diesem Programm die Funktion `void mmul(...)` und schreiben Sie eine entsprechende Funktion in einer separaten Datei in Assembler.

Linken Sie anschließend die Assembler-Funktion mit dem verbleibenden C-Programm zu einer Binary und überprüfen Sie damit die korrekte Funktionsweise Ihrer Assembler-Routine. Angenommen, Ihre C-Quellcodedatei heißt `matrix.c` und Ihre Assembler-Quellcodedatei heißt `mmul.asm`, so können Sie die beiden Dateien folgendermaßen kompilieren/assemblieren und linken:

```
as -o mmul.o mmul.asm
gcc -o matrix mmul.o matrix.c
```

Es steht Ihnen frei, auf 32- oder 64-Bit-Basis zu arbeiten. Verwenden Sie bei einem 32-Bit-Programm die Aufrufkonvention cdecl, bei einem 64-Bit-Programm dagegen die Aufrufkonvention AMD64 ABI.

Hinweise: Beachten Sie, dass neben Werten auch Pointer beim Funktionsaufruf übergeben werden. Bei diesen Pointern auf die Matrizen handelt es sich um Werte, die Sie direkt als Basis für eine indirekte Adressierung verwenden können. Bedenken Sie dabei auch, dass die Adressierung auf Grundlage von Bytes erfolgt.

Aufgabe 33: Aufruf von glibc-Funktionen in Assembler (4 Punkte)

In der vorangegangenen Aufgabe haben Sie eine Assembler-Funktion aus einem C-Programm heraus aufgerufen. In dieser Aufgabe sollen Sie – praktisch umgekehrt – Funktionen der C-Standardbibliothek *glibc* aus einem Assembler-Programm heraus aufrufen.

Schreiben Sie ein Assembler-Programm, das folgendes leistet:

- Ein maximal 20 Zeichen langer String wird von der Standardeingabe `stdin` mithilfe der glibc-Funktion `fgets` eingelesen.
- Der String wird komplett in Großbuchstaben umgewandelt (das heißt kleine Buchstaben werden in große umgewandelt, alle anderen Zeichen bleiben unverändert).
- Falls der String einen Zeilenumbruch enthält, soll dieser nicht mit ausgegeben werden.
- Der umgewandelte String wird mithilfe der glibc-Funktion `puts` auf der Standardausgabe `stdout` ausgegeben.

Sie können Ihr Programm in 32 oder 64 Bit kompilieren. Finden Sie heraus, wie Sie es mit der glibc linken müssen!

Tipp: Schreiben Sie zuerst ein C-Programm, das die obigen Anforderungen erfüllt. Dieses müssen Sie dann „nur noch“ nach Assembler portieren.

Hinweise:

- *Beachten Sie das terminierende Nullbyte für den String!*
- *Die 32-Bit-glibc verwendet als Aufrufkonvention cdecl, die 64-Bit-glibc hingegen AMD64 ABI.*
- *Für den Aufruf von `fgets` benötigen Sie den File Descriptor `stdin`. Diesen können Sie in Ihrem Assembler-Programm genau wie die benötigten glibc-Funktionen einfach ohne explizite Deklaration bzw. Inklusion verwenden, wenn Sie mit der glibc linken.*

Aufgabe 34: Reverse Engineering von Shellcode (4 Punkte + 1 Bonuspunkt)

Der Administrator eines Webhosting-Unternehmens wendet sich an Sie, weil Sie sich mit x86-Assembler auskennen und ihm bei der Untersuchung eines Sicherheitsvorfalls helfen sollen. Das Network Intrusion Detection System (NIDS) des Unternehmens hat die Entdeckung von potenziellem Shellcode in einem Paket gemeldet. Ziel des potenziellen Angriffs war ein 32-Bit x86 Linux-Server, der in einem vom Internet aus erreichbaren Dienst eine Verwundbarkeit für einen Buffer-Overflow-Exploit aufwies.

Der Administrator konnte aus dem geloggten Paket bereits den Teil extrahieren, der potenziellen Maschinencode enthält (diesen finden Sie auch auf der Vorlesungshomepage zum Download):

```
00000000: 31 db f7 e3 53 43 53 6a 02 89 e1 b0 66 cd 80 5b
00000010: 5e 52 68 ff 02 04 d2 6a 10 51 50 89 e1 6a 66 58
00000020: cd 80 89 41 04 b3 04 b0 66 cd 80 43 b0 66 cd 80
00000030: 93 59 6a 3f 58 cd 80 49 79 f8 68 2f 2f 73 68 68
00000040: 2f 62 69 6e 89 e3 50 53 89 e1 b0 0b cd 80
```

a)

Disassemblieren Sie den Maschinencode! Sie können dazu z.B. einen Online-Disassembler wie auf `http://www.onlinedisassembler.com/odaweb/run_hex` verwenden.
Zur Kontrolle: Der korrekte Code beginnt mit `xorl %ebx, %ebx`. Fügen Sie dann sinnvolle Kommentare hinzu! Zum Verständnis der verwendeten Syscalls kann z.B. folgende Seite als Einstieg dienen:
`http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html`.

b)

Was tut der Code? Geben Sie möglichst genau an, welche Funktionen mit welchen Parametern aufgerufen werden und was dadurch passiert. Sie können statt einer reinen Beschreibung auch ein äquivalentes, gut kommentiertes C-Programm schreiben.

c)

Dieser Aufgabenteil wird mit einem Bonuspunkt bewertet!

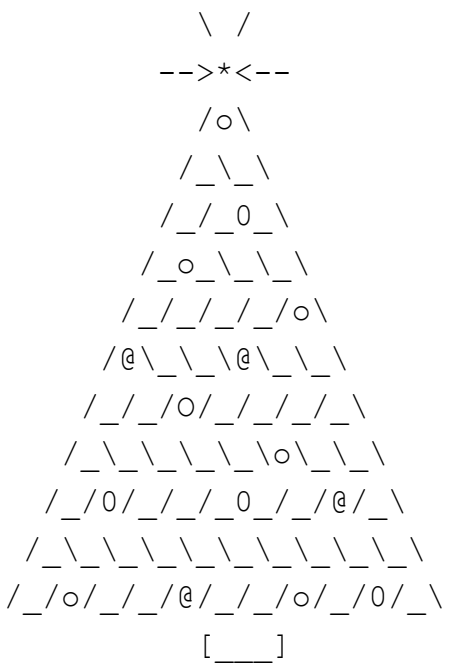
Welche spezielle Einschränkung gilt für Shellcode von Exploits, die Buffer Overflows von in C geschriebenen Programmen ausnutzen?

Aufgabe 35: Weihnachts-Bonus-Aufgabe (max. 4 BONUS-Punkte = 2 + 1 + 1)

Schreiben Sie ein C-Programm, das einen Weihnachtsbaum in ASCII-Art auf der Kommandozeile ausgibt.

Dabei sollen als Kommandozeilenparameter die Höhe des Baumes und die Anzahl der Kugeln übergeben werden. Die Breite des Baums soll entsprechend der angegebenen Höhe skaliert werden. Die Kugeln sollen zufällig in dem Baum platziert werden.

Ihr Baum könnte zum Beispiel wie folgt aussehen:



a) Sie erhalten 2 Bonus-Punkte, wenn die Aufgabe bis Fr. 09.01.2015 gelöst und ins SVN-Repository eingespielt wird.

b) Sie erhalten 1 weiteren Bonus-Punkt, wenn die Aufgabe **bereits bis Heiligabend, Mi. 24.12.2014 um 23.59 Uhr** gelöst und ins SVN-Repository eingespielt wird – **und bis dahin zwei Lösungen** (ASCII-Ausgaben) für 2 verschiedene Parametersets **an die Sys-Prog-Mailingliste vl-sys-prog@lists.iai.uni-bonn.de geschickt** werden.

c) Sie erhalten 1 weiteren Bonus-Punkt, wenn die Aufgabe anstatt als C-Programm **in Assembler gelöst wurde** (als Basis könnte das Assembler-Programm-Beispiel „stars“ der Vorlesung dienen – der main()-Teil darf in C bleiben, vgl. Folie 66).

Wir wünschen Ihnen fröhliche Festtage und ein erfolgreiches Neues Jahr 2015!