

Buffer Overflows

Valentin Brandl <mail@vbrandl.net>

Fakultät Informatik und Mathematik

21. September 2022

1. Problem

2. Beispiel

3. Stack Layout, Execution Flow

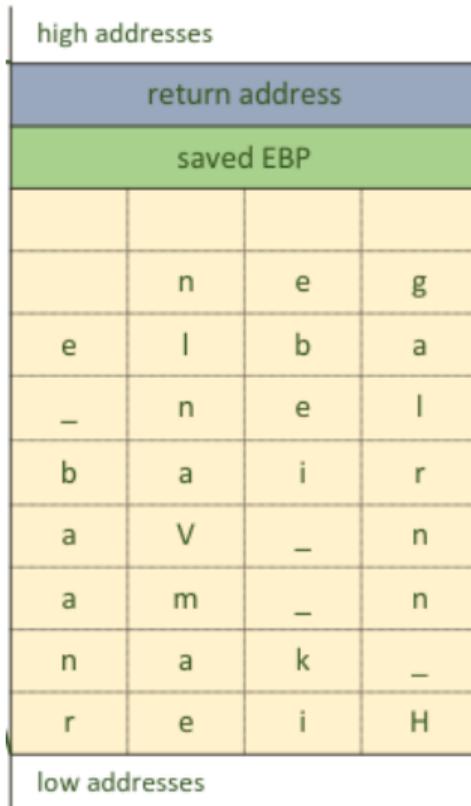
4. Exkurs: Shellcode

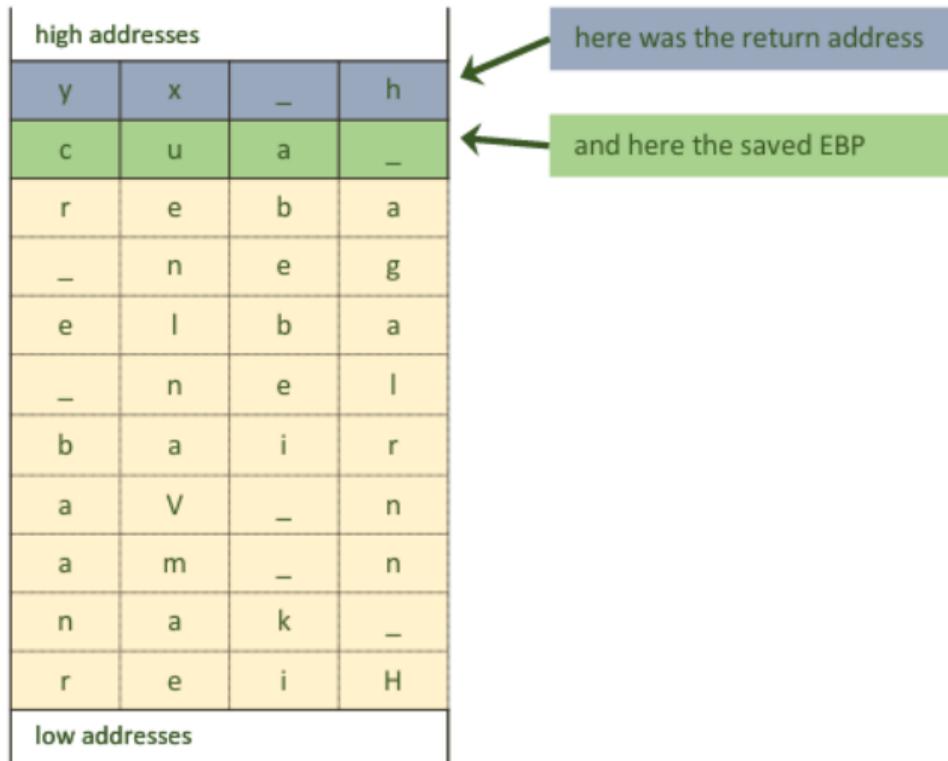
5. Exploitation

- ▶ Maschinennahe Programmiersprachen ohne Memorysafety (z.B. C, C++, Assembly, FORTRAN) erlauben es, Speicher beliebig zu beschreiben
- ▶ Bei fehlender Validierung kann ein Programm mehr Speicher schreiben, als eigentlich reserviert wurde und dabei andere Daten im RAM überschreiben
- ▶ Entsprechend präparierter Input kann dazu führen, dass ein Angreifer den Ablauf der Programmausführung übernehmen kann

```
void foo(char *input) {  
    int is_logged_in = 0;  
    char buf[50];  
    strcpy(buf, input);  
    if (is_logged_in) {  
        puts("logged in!!1!");  
    } else {  
        puts("not logged in");  
    }  
}
```

```
void foo(char *input) {  
    char buf[50];  
    strcpy(buf, input);  
    puts(buf);  
}
```





- ▶ Beim Aufruf einer Funktion, aktuelle Adresse auf Stack
- ▶ *Instruction Pointer (IP)* auf Adresse der aufgerufenen Funktion
- ▶ Bei **return**, Stack Frame wiederherstellen, Adresse von Stack in *IP*
- ▶ Was wenn die Adresse auf dem Stack überschrieben wurde?

- ▶ Shellcode ist der Maschinencode, der nach Übernahme des Ausführungsablauf ausgeführt werden soll
- ▶ Buffer kann klein sein \Rightarrow Shellcode häufig auf Größe optimiert
- ▶ Häufig Strings als Eingabe
- ▶ In C terminiert mit `\0` \Rightarrow Payload darf kein `0x00` enthalten, da alles danach abgeschnitten wird
- ▶ \Rightarrow Selbst schreiben ist möglich, aber aufwändig und setzt Kenntnisse in Assembly und der anzugreifenden Architektur/OS/... voraus
- ▶ Verfügbare, öffentliche Sammlungen verwenden:
 - ▶ <https://shell-storm.org/shellcode/>
 - ▶ <https://www.exploit-db.com/shellcodes>

- ▶ Shellcode im Speicher plazieren
- ▶ Buffer überschreiben
- ▶ *IP* überschreiben
- ▶ *IP* auf Shellcode zeigen lassen

- ▶ Address Space Layout Randomization (ASLR)
- ▶ w^x Memory
- ▶ Runtime Bounds Checks
- ▶ Typesystem basierte Lösungen [Con07]

[Con07]

Condit, Jeremy and Harren, Matthew and Anderson, Zachary and Gay, David and Necula, George C. “Dependent Types for Low-Level Programming”. In: *Programming Languages and Systems*. 2007.