

# Buffer Overflows

Valentin Brandl <sup>1</sup>

Fakultät Informatik und Mathematik

<sup>1</sup><mail@vbrandl.net>

5. Oktober 2022

1. Problem

2. Beispiel

3. Stack Layout, Execution Flow

4. Exkurs: Shellcode

5. Exploitation

# 1. Problem

- ▶ Maschinennahe Programmiersprachen ohne Memorysafety (z.B. C, C++, Assembly, FORTRAN) erlauben es, Speicher beliebig zu beschreiben  
 $(arr[i] == arr + \text{sizeof}(\text{int}) * i)$
- ▶ 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

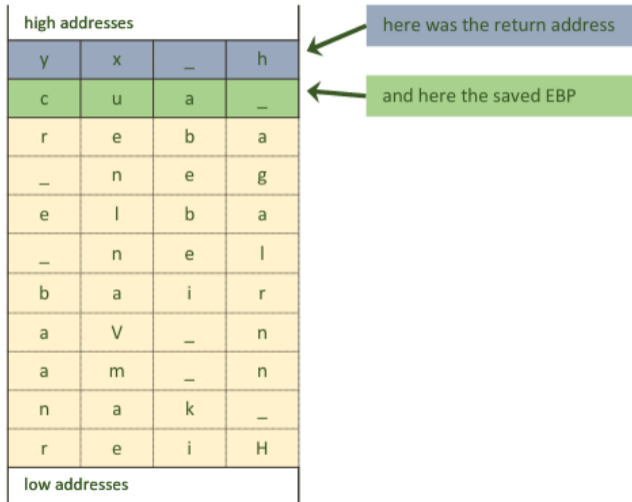
## 2. Beispiel

```
void foo(char *input) {  
    int is_logged_in = 0;  
    char buf[64];  
    strcpy(buf, input);  
    if (is_logged_in) {  
        puts("logged in!!1!");  
    } else {  
        puts("not logged in");  
    }  
}
```

## 3. Stack Layout, Execution Flow







- ▶ 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?

## 4. Exkurs: Shellcode

- ▶ 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>

## 5. Exploitation

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

## 6. Aktuelles Beispiel

*Chromium* Heap Buffer Overflown WebGPU:  
*CVE-2022-1483* [22a]

*Linux Kernel* Heap Buffer Overflow durch Integer  
Overflow: *CVE-2022-39842* [22e]

*MPlayer* Buffer Overflow beim lesen von AVI und MPEG  
Headern: *CVE-2022-38866* [22d],  
*CVE-2022-38864* [22c]

*GNU Binutils* Heap Buffer Overflow in *strip*:  
*CVE-2022-38533* [22b]



## 7. Aktuelle Situation

- ▶ Address Space Layout Randomization (ASLR)
- ▶  $w^*x$  Memory
- ▶ Runtime Bounds Checks
- ▶ Typesystem basierte Lösungen [Con+07]

[22a]

*CVE-2022-1483*. Available from MITRE,  
CVE-ID CVE-2022-1483. Apr. 2022. URL:  
[https://cve.mitre.org/cgi-  
bin/cvename.cgi?name=CVE-2022-1483](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-1483)  
(besucht am 28.09.2022).

[22b] *CVE-2022-38533*. Available from MITRE, CVE-ID CVE-2022-38533. Aug. 2022. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-38533> (besucht am 28.09.2022).

[22c] *CVE-2022-38864*. Available from MITRE, CVE-ID CVE-2022-38864. Aug. 2022. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-38864> (besucht am 28.09.2022).

- [22d] *CVE-2022-38866*. Available from MITRE, CVE-ID CVE-2022-38866. Aug. 2022. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-38866> (besucht am 28.09.2022).
- [22e] *CVE-2022-39842*. Available from MITRE, CVE-ID CVE-2022-39842. Aug. 2022. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-39842> (besucht am 28.09.2022).

[Con+07]

Jeremy Condit u. a. “Dependent Types for Low-Level Programming”. In: *Programming Languages and Systems*. 2007.