Weiterbildung gibb

HF Informatik & Telekommunikation
Höhere Fachschule

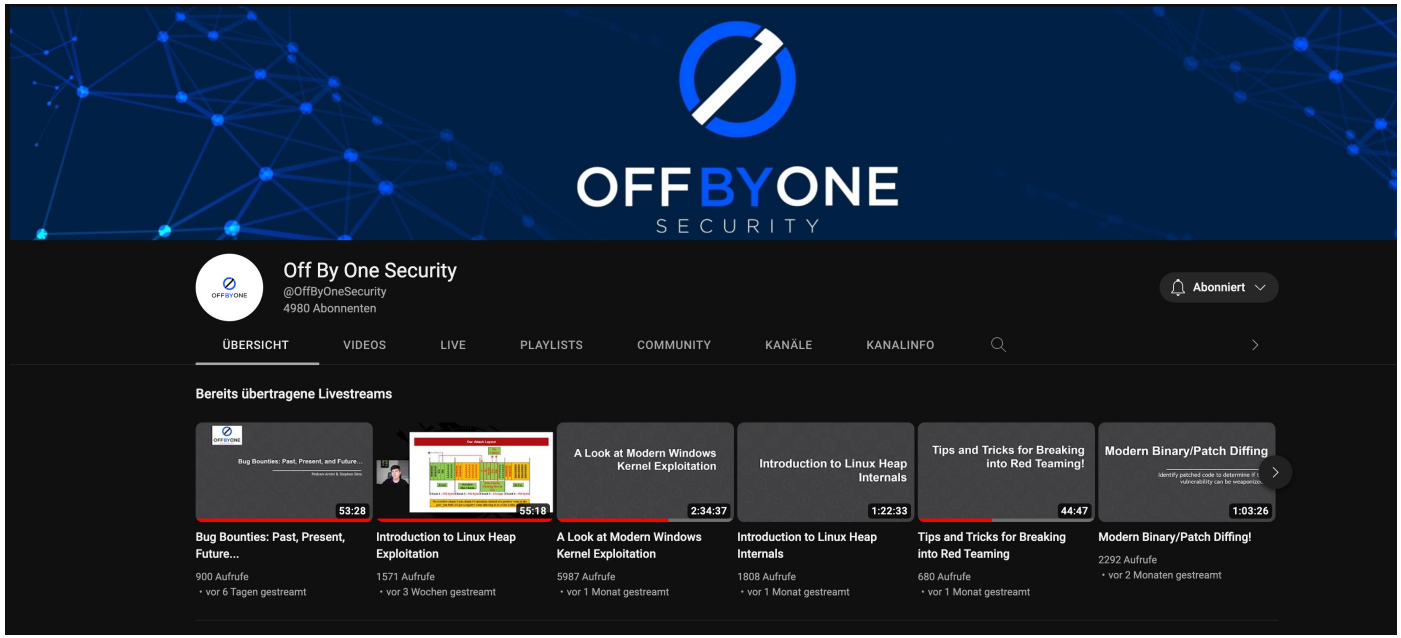
Cybersecurity II

Buffer Overflow
Dec 2022

CYS II – Cyber Security II
Ralph Maurer

Objectives

- Build an abstract imagination of a buffer overflow based code execution.



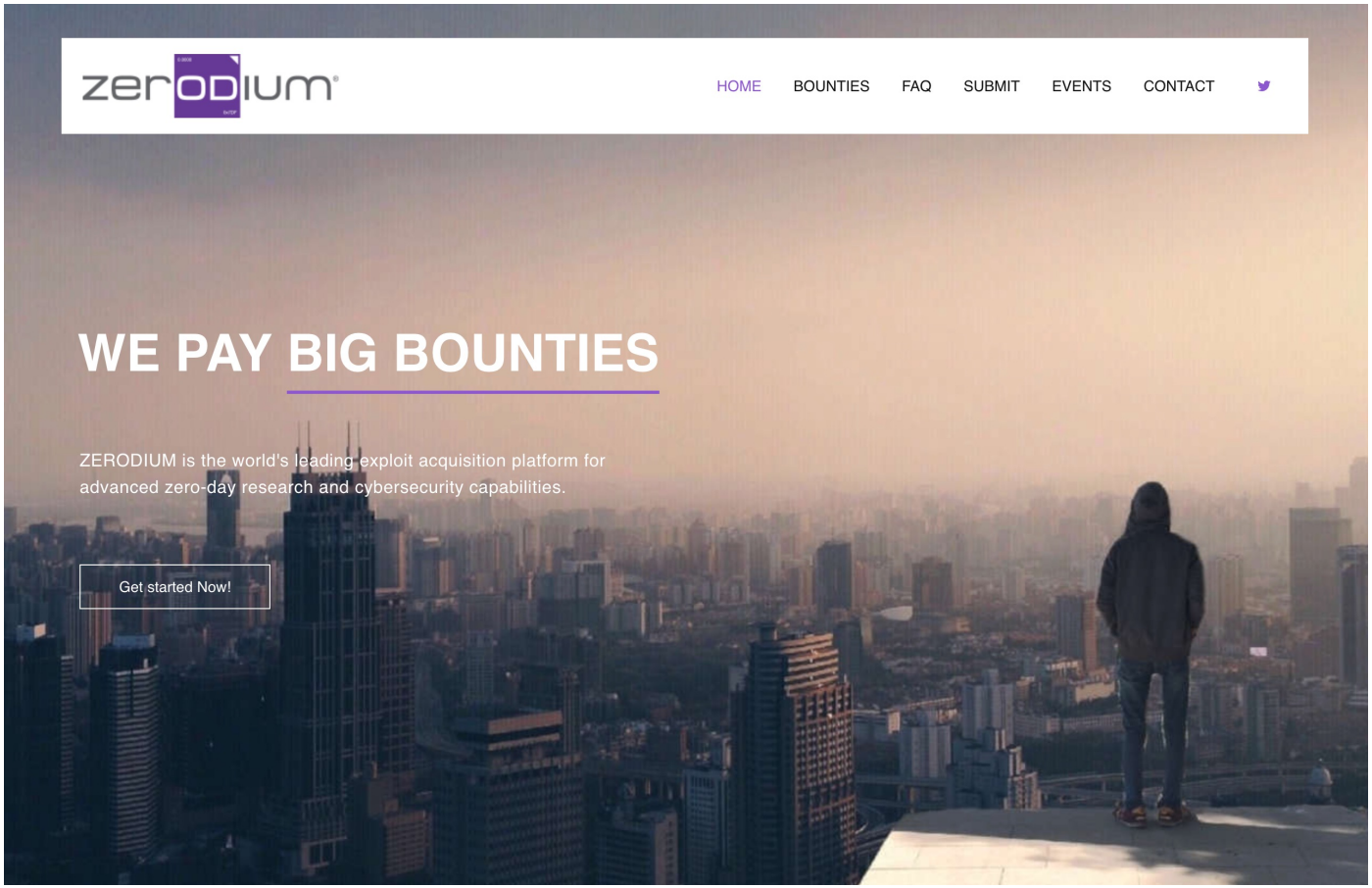
https://www.youtube.com/@OffByOneSecurity

Real stuff

https://www.corelan.be/

zerodium

- Money for Hacking



https://zerodium.com/

Agenda

- Buffer Overflow
- Simplified Computing Process
- Memory Layout
- Stack Architecture
- Stack Overflow
- Heap Overflow
- Live Demo (Video)

Buffer Overflow

Buffer Overflow

- Exploiting a buffer overflow is a well-known security exploit – affected by Buffer Overflow attacks:
- Firefox & Tor Browser
  https://www.heise.de/security/meldung/Schlupfloecher-fuer-Angreifer-in-Firefox-und-Tor-Browser-gestopft-3949836.html
- Adobe Acrobat Reader
  https://nvd.nist.gov/vuln/detail/CVE-2018-15951
- Windows Server attacked by EternalBlue (WannaCry RansomWare)
  https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2017-0144


Schlupflöcher für Angreifer in Firefox und Tor Browser gestopft

Die Entwickler haben zum Teil kritische Lücken in Firefox und Firefox ESR geschlossen. Davon profitiert auch der auf Anonymität getrimmte Tor Browser.

Lesezeit: 2 Min. In Pocket speichern 28

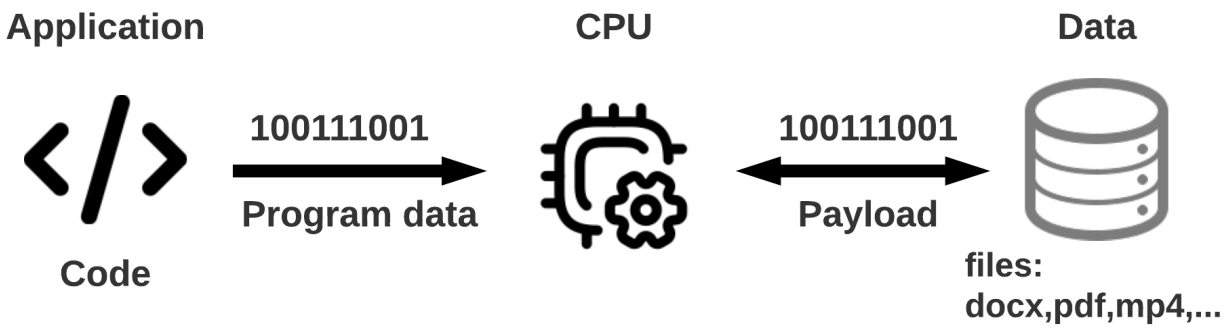
(Bild: pixabay)

24.01.2018 10:42 Uhr | Security

Von Dennis Schirrmacher

Simplified Computing Process

- **CPU** is the heart and/or the brain of a computer
- **CPU** executes the instructions (code) that are provided to it
- **CPU** can get Programm data input to run instructions
- **CPU** can send and recieve data from memory/storage – like user data (input/output I/O)
- In **computing** the **payload** is the part of **transmitted data** between two partner like for example the **CPU** and the **Data**

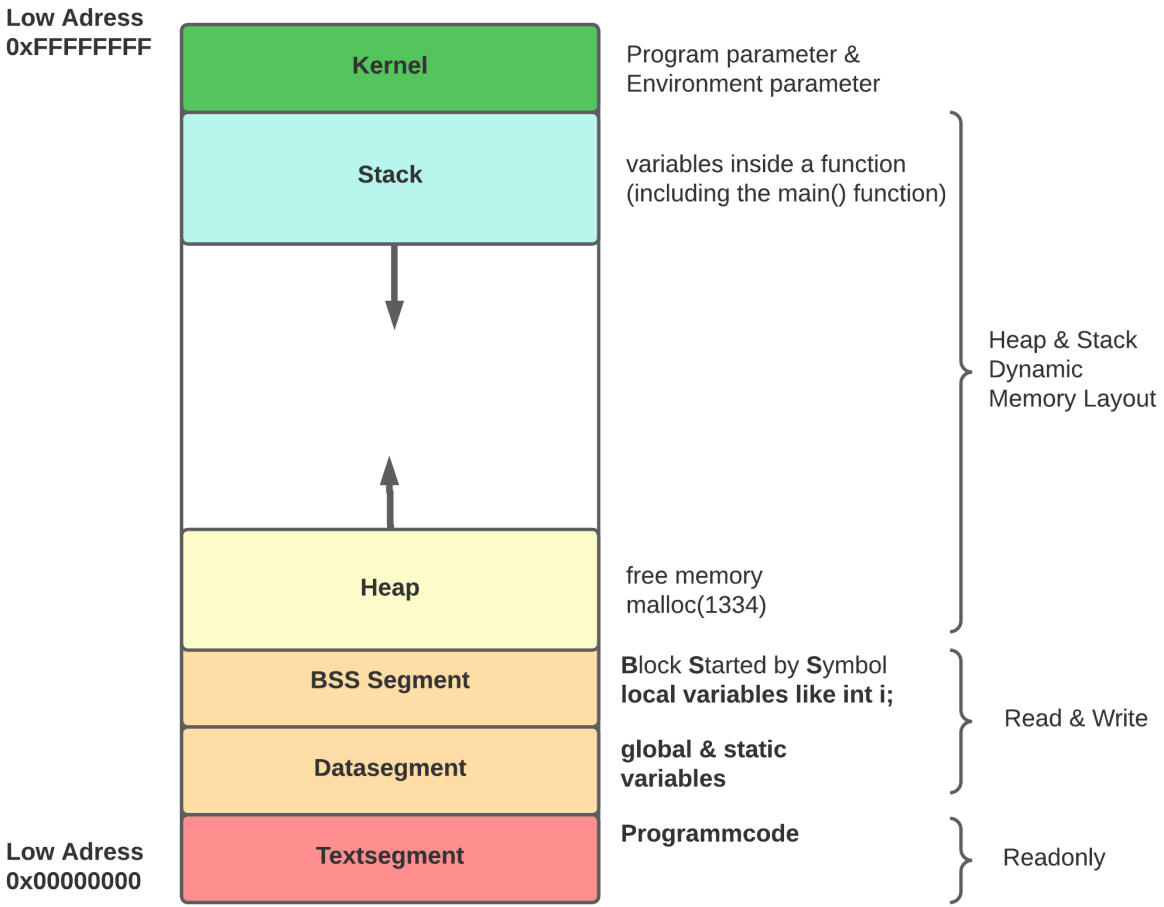


How does a cpu works:

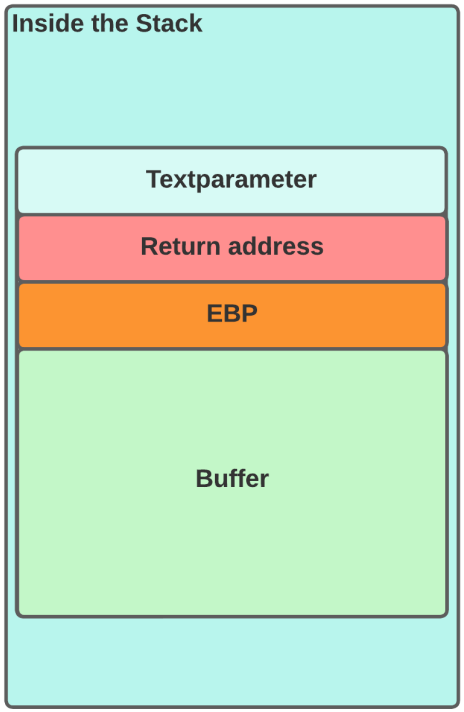
https://www.freecodecamp.org/news/how-does-a-cpu-work/

Memory Layout

Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011



Stack Architecture for a C-Programm

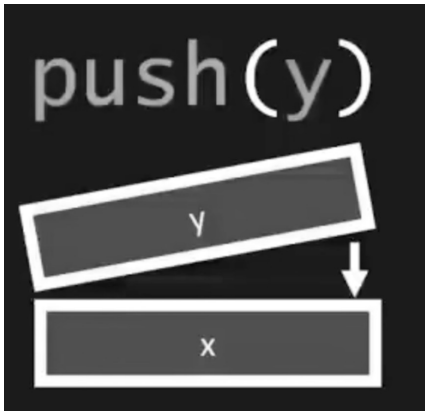


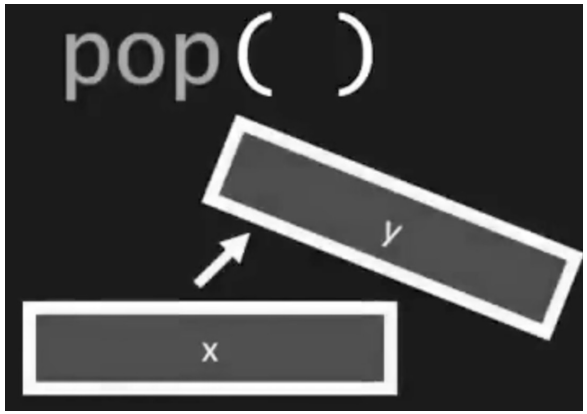
Where to go after execution

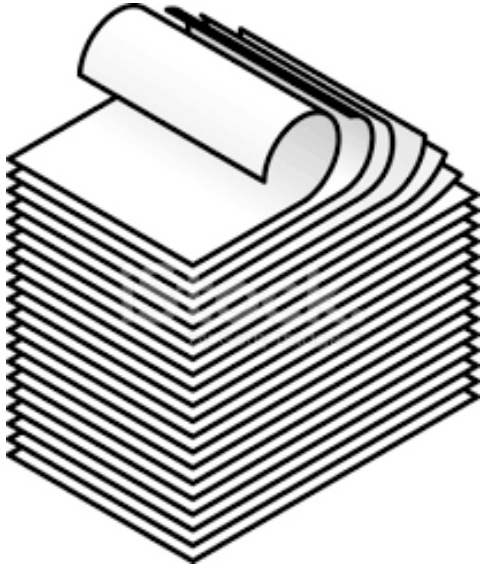
Extented Base Pointer
The base pointer is the top of the buffer

Stack commands

- **Stack** works in LIFO order (Last in – First out)
- **push(y)** brings y on the stack
- **pop()** removes last in







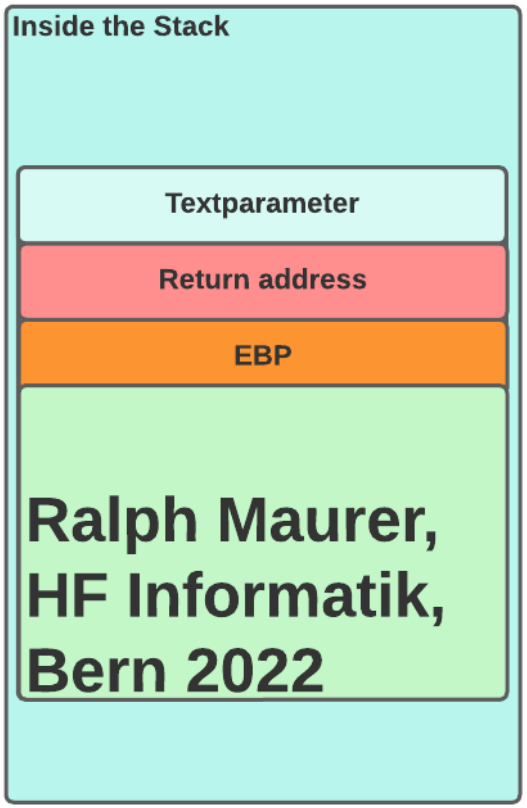
Stack overflow

vuln.c

```c
#include<stdio.h>
#include<string.h>

// print function for a text
void echo(char* text){
    // reserve buffer for 50 characters
    char buf[50];
    // copy text to buffer
    strcpy(buf, text);
    // print buffer
    printf("%s\n", buf);
}

int main (int argc, char* argv[]){
    // !!! no length check !!!
    // if argv[1] is longer than 50, segment fault (Memory violation)
    echo(argv[1]);
    return 0;
}
```

Stack overflow

```
┌──(vmadmin㉿vmkl1)-[~/workspace/BufferOverflow]
└─$ gcc vuln.c -o vuln

┌──(vmadmin㉿vmkl1)-[~/workspace/BufferOverflow]
└─$ ./vuln "Ralph Maurer, HF Informatik, Bern 2023"
Ralph Maurer, HF Informatik, Bern 2022
```



Stack overflow

```
┌──(vmad./vuln
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
zsh: segmentation fault  ./vuln
```

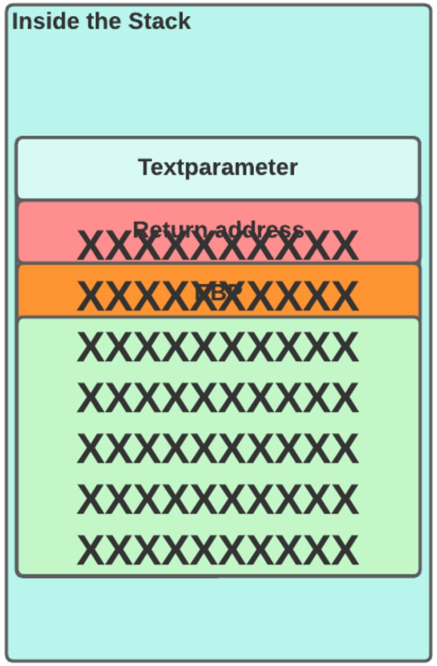
Bad coding creates opportunities for criminals

From Bug To The Exploit

An attacker can use the **Return address** to execute a malicious code
The Return address points on the **next instruction** for the CPU.

Let's do that.

Instead of using a malicious code we start the firefox browser with
http://www.gibb.ch/weiterbildung/informatik
and we shutdown the Kali Linux



Heap overflow: Some easy C-Code for trivial App

vuln_ce.c

```c
#include<stdio.h>
#include<stdlib.h>

int main (int argc, char* argv[]){
    // reserve 12 characters on the heap with malloc
    char * name = (char*) malloc(12);
    // reserve 60 characters on the heap
    char * bad_command = (char*) malloc(60);
    // print the memoryaddress for name and bad_command
    // The difference between the two memoryaddress is the size
    // of the memory buffer
    printf("Memoryaddress of *Name*: <%d>\n", name);
    printf("Memoryaddress of *bad_command*: <%d>\n", bad_command);

    printf("What is your name?");
    // write input to heap - you should never use gets!!!
    // input >32 provokes the Heap Buffer Overflow
    gets(name);
    printf("Hello %s! So nice to have you here :P \n", name);
    // run a command ;)
    system(bad_command);
    return 0;
}
```

Heap overflow

```
┌──(vmadmin㉿vmkl1)-[~/workspace/BufferOverflow]
└─$ ./vuln_ce
Memoryaddress of *Name*: <-1092803936>
Memoryaddress of *bad_command*: <-1092803904>
What is your name?Ralph
Hello Ralph! So nice to have you here :P
```

1092803936 - 1092803904 = 32
Buffersize for 32 characters

```
┌──(vmadmin㉿vmkl1)-[~/workspace/BufferOverflow]
└─$ ./vuln_ce
Memoryaddress of *Name*: <532505248>
Memoryaddress of *bad_command*: <532505280>
What is your name?12345678901234567890123456789012firefox www.gibb.ch/Weiterbildung/Informatik
                  \__________32 character_______/\__________could be malicious command_______/
```

```
┌──(vmadmin㉿vmkl1)-[~/workspace/BufferOverflow]
└─$ ./vuln_ce
Memoryaddress of *Name*: <767499248>
Memoryaddress of *bad_command <767499280>
What is your name?12345678901234567890123456789012shutdown -h now
```

Live-Demo