EGGHUNTING WORKSHOP

IT SECURITY MEETUP KASSEL / NORDHESSEN

DENNIS KNIEP - 27.05.2017


EGG HUNTERS
Welcome
Holiday Inn

AGENDA

0x01 Buffer Overflow
0x02 **Egghunting**
0x03 Linux (Egghunting)
0x04 Fuzzing
0x05 Windows (Fuzzing & Egghunting)

BUFFER OVERFLOW

BASICS

MEAT BEFORE MILK

PROGRAM EXECUTION

- CPU
- Memory

CPU

CPU loads instruction (code) from memory and executes it

CPU INSTRUCTIONS

- load information from memory into registers (mov)
- process data from memory or registers (comp, add, sub)
- change next executed instruction (jmp, call, int)
- etc.

CPU REGISTERS

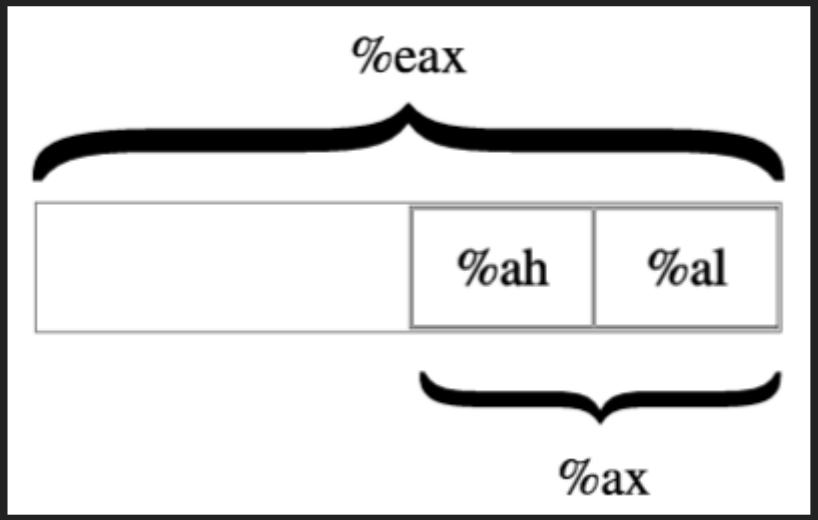
Most information is stored in memory, brought in to the registers for processing, and then put back into memory when the processing is completed.

Registers are what the computer uses for computation.

GENERAL-PURPOSE REGISTERS

- eax
- ebx
- ecx
- edx
- edi
- esi

REGISTERS SIZE



4 bytes / 32 bit

0xFFFFFFFF

SPECIAL-PURPOSE REGISTERS

- ebp - base pointer register
- esp - address of the value on the top of the stack
- eip - instruction pointer
- eflags

CPU CYCLE WITH EIP

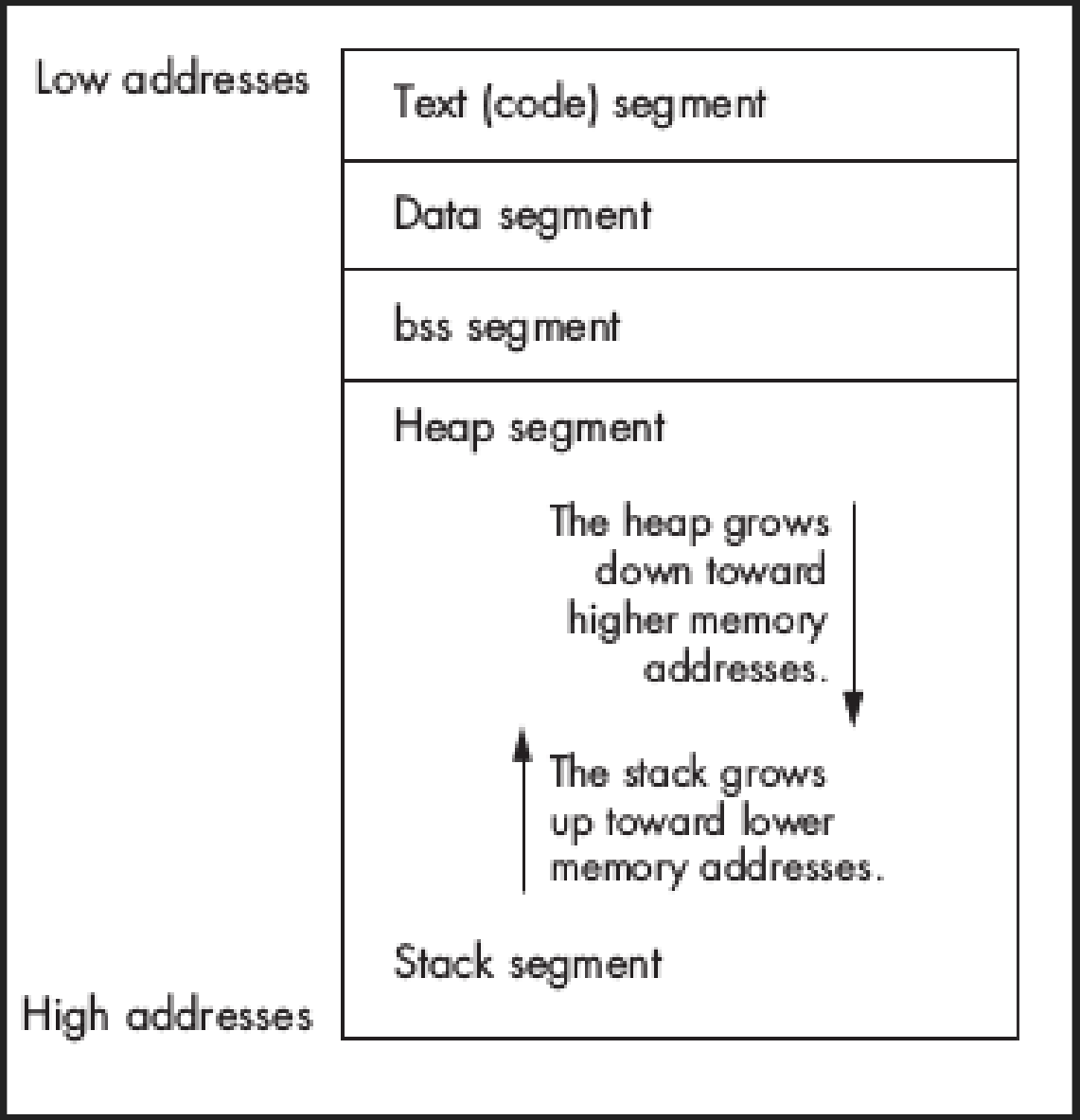
- EIP points to next instruction in memory
- CPU loads OpCode from memory
- Increments EIP
- Executes OpCode

MEMORY

0X0 - 0XFFFFFFFF

Jeder Prozess arbeitet virtuell mit dem gesamten Memory

A COMPILED PROGRAM'S MEMORY



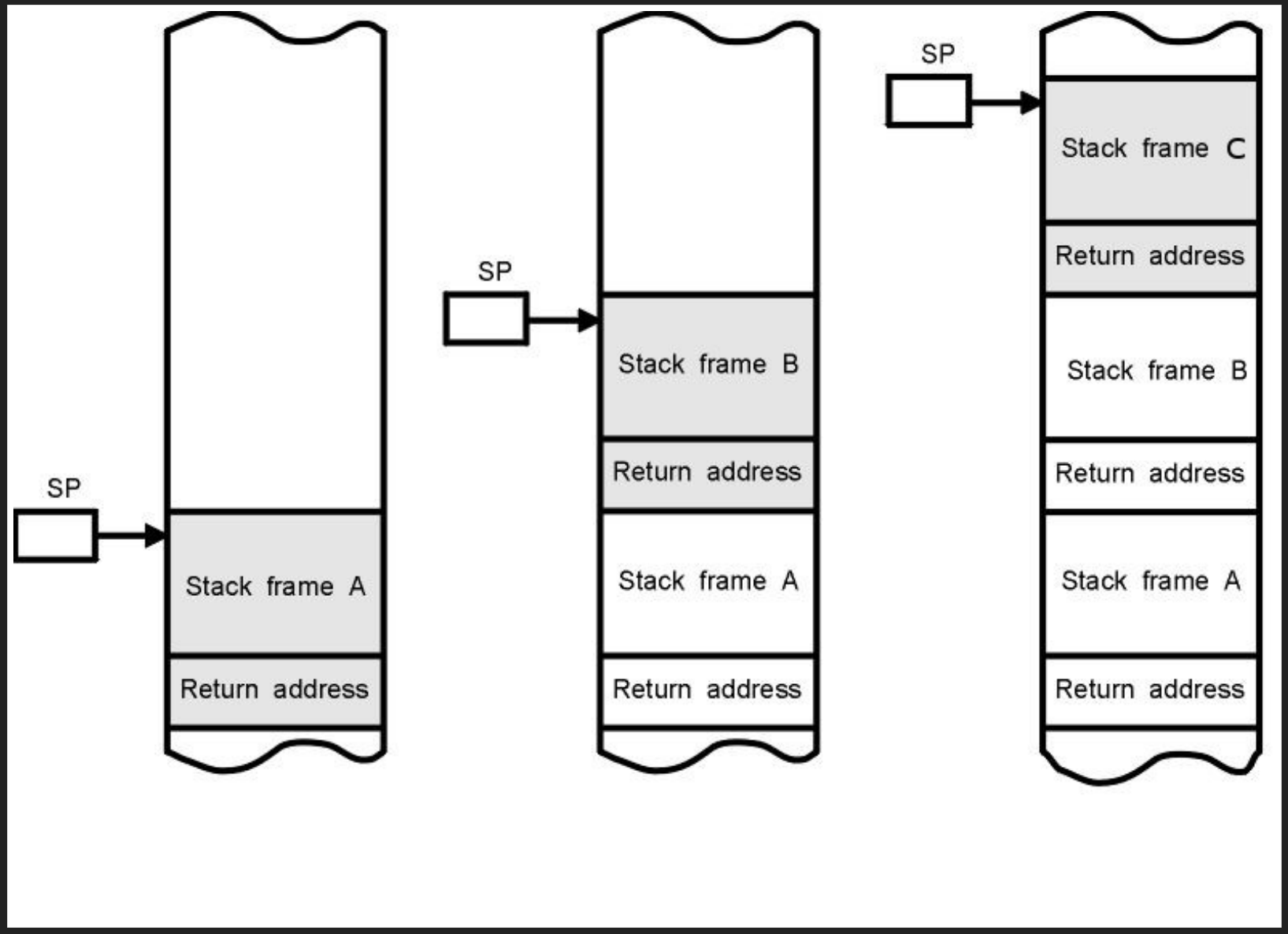
STACK

Used for storeing the data for the current executed function chain.

STACKFRAMES

```
void C(){

}

int B(){
        C();
        return 2;
}

int A(int x){
        int b = B();
        return x + b;
}

int main(int argCount, char *args[]){
        return A(15);
}
```

STACKFRAMES



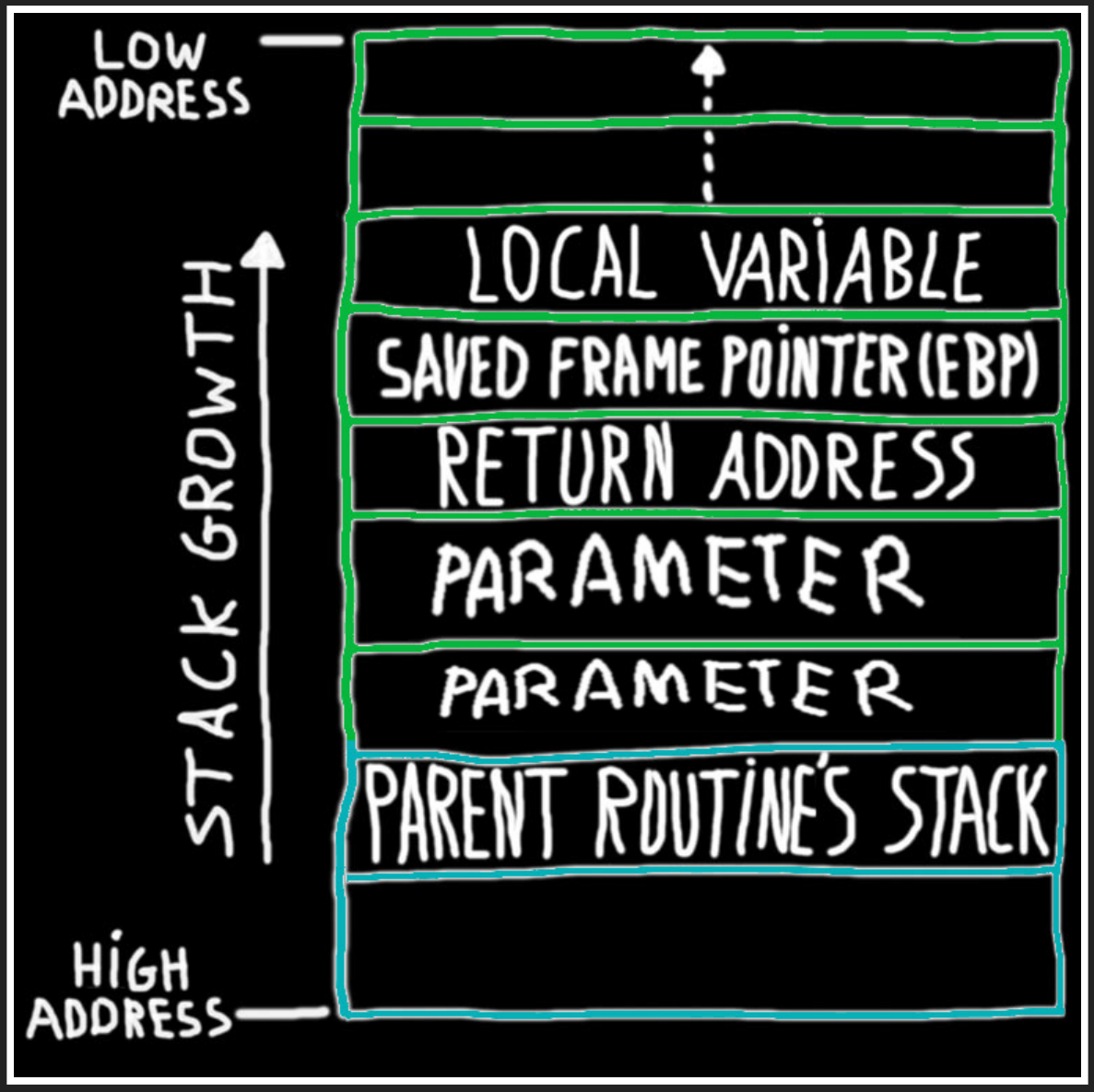
FUNKTIONSAUFRUF SCHEMA

```
A(15);
```

- (Save registers to stack)
- Add Functionparameter to Stack
- Call the Function
  (Stores Return EIP, Set EIP to Function)

- Save calling functions ebp
- Set current functions ebp

STACKFRAME


LOW ADDRESS
STACK GROWTH
HIGH ADDRESS
LOCAL VARIABLE
SAVED FRAME POINTER (EBP)
RETURN ADDRESS
PARAMETER
PARAMETER
PARENT ROUTINE'S STACK

BASEPOINTER

constant reference to the stack frame

Access all of the data in the function by using base pointer addressing using different offsets from ebp!

- Local Var 2: [ebp-8]
- Local Var 1: [ebp-4]
- Parameter 1: [ebp+8]
- Parameter 2: [ebp+12]

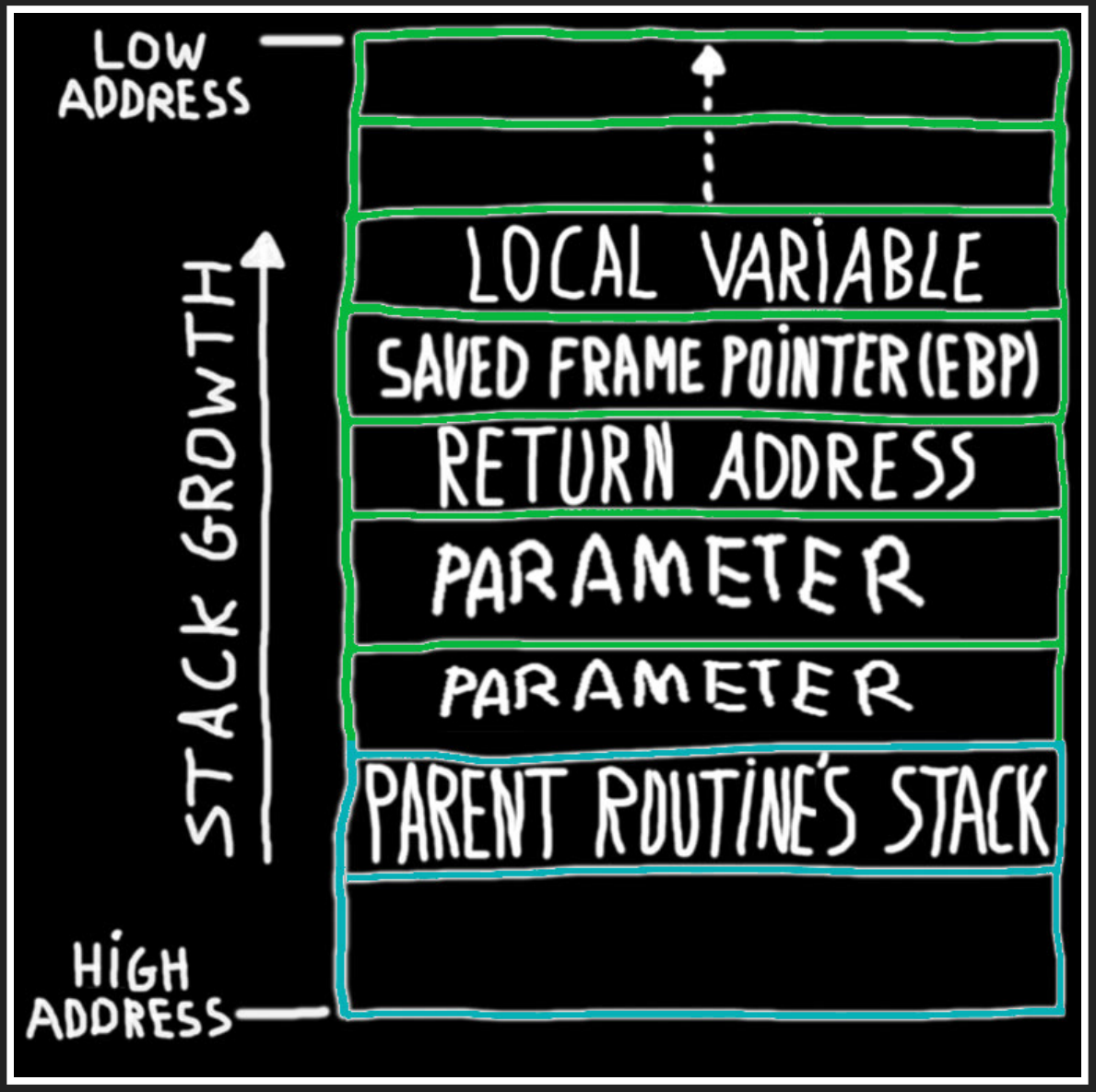
FUNKTIONSABBAU SCHEMA

```
return;
```

- Store return value (in eax)
- Reset Stack to current ebp
- Restore calling functions ebp
- Leave Function (Set EIP to Return EIP)

- Remove the Parameters from Stack
- (Pop the saved registers from the Stack)
- Process return value

STACKFRAME


LOW ADDRESS
STACK GROWTH
HIGH ADDRESS
LOCAL VARIABLE
SAVED FRAME POINTER (EBP)
RETURN ADDRESS
PARAMETER
PARAMETER
PARENT ROUTINE'S STACK

STACKFRAMES

KONVENTION, KEINE PFLICHT FÜR CPU

- Compiler übersetzt den C Code so, dass er Stackframes benutzt
- Stack könnte aber aus Sicht der CPU völlig anders genutzt werden
- Es gibt keinen Supervisor

Wiki Calling Convention

LOKALE VARIABLEN

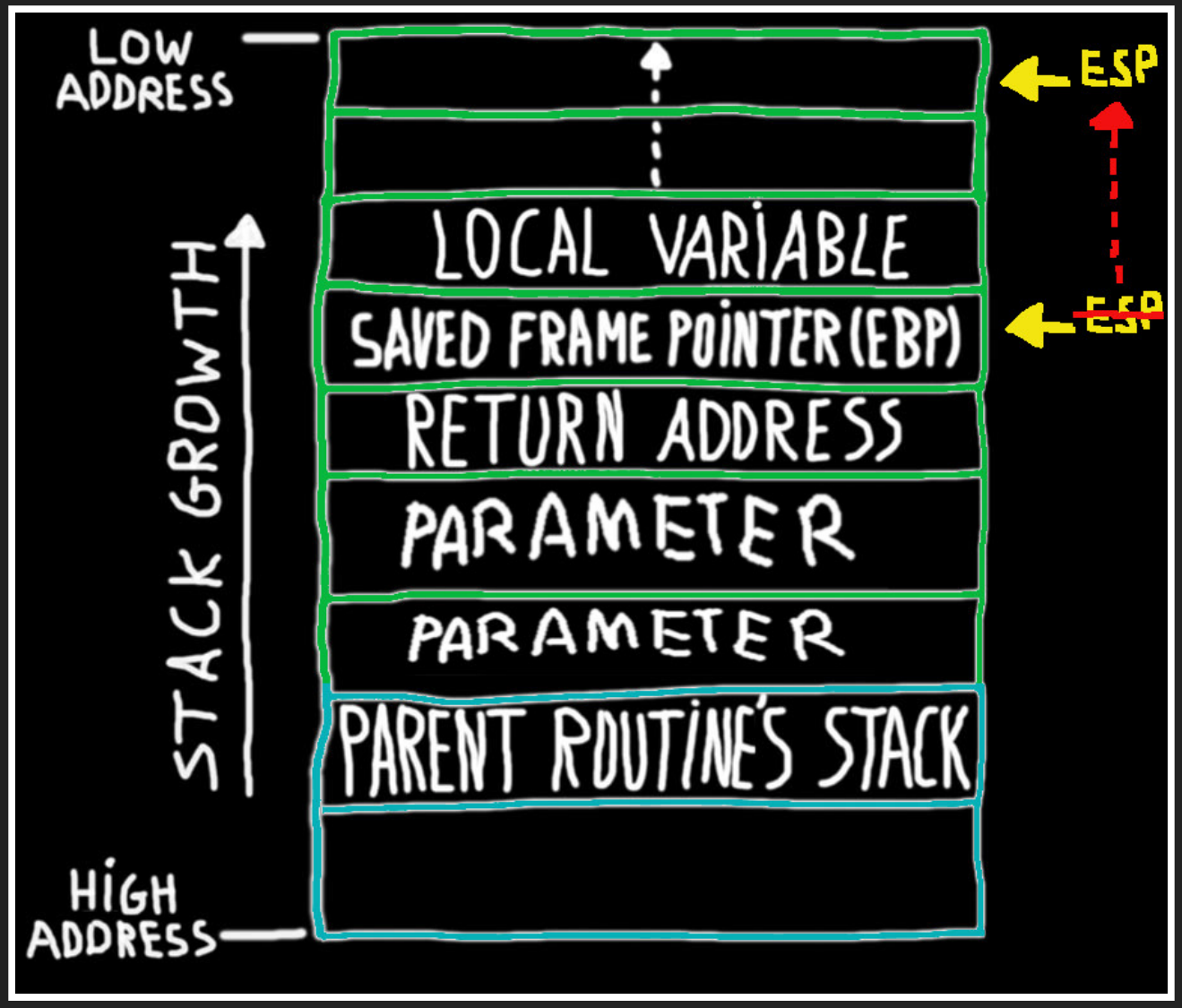
Lokale Variablen werden im Stackframe gespeichert.

```
int main(int argCount, char *args[]){
        char input[100];
        return 0;
}
```

```
sub     esp,0x70
```

LOKALE VARIABLEN

Platz für lokale Variable geschaffen.



LOKALE VARIABLEN

Es wurde Platz für 100 byte geschaffen.

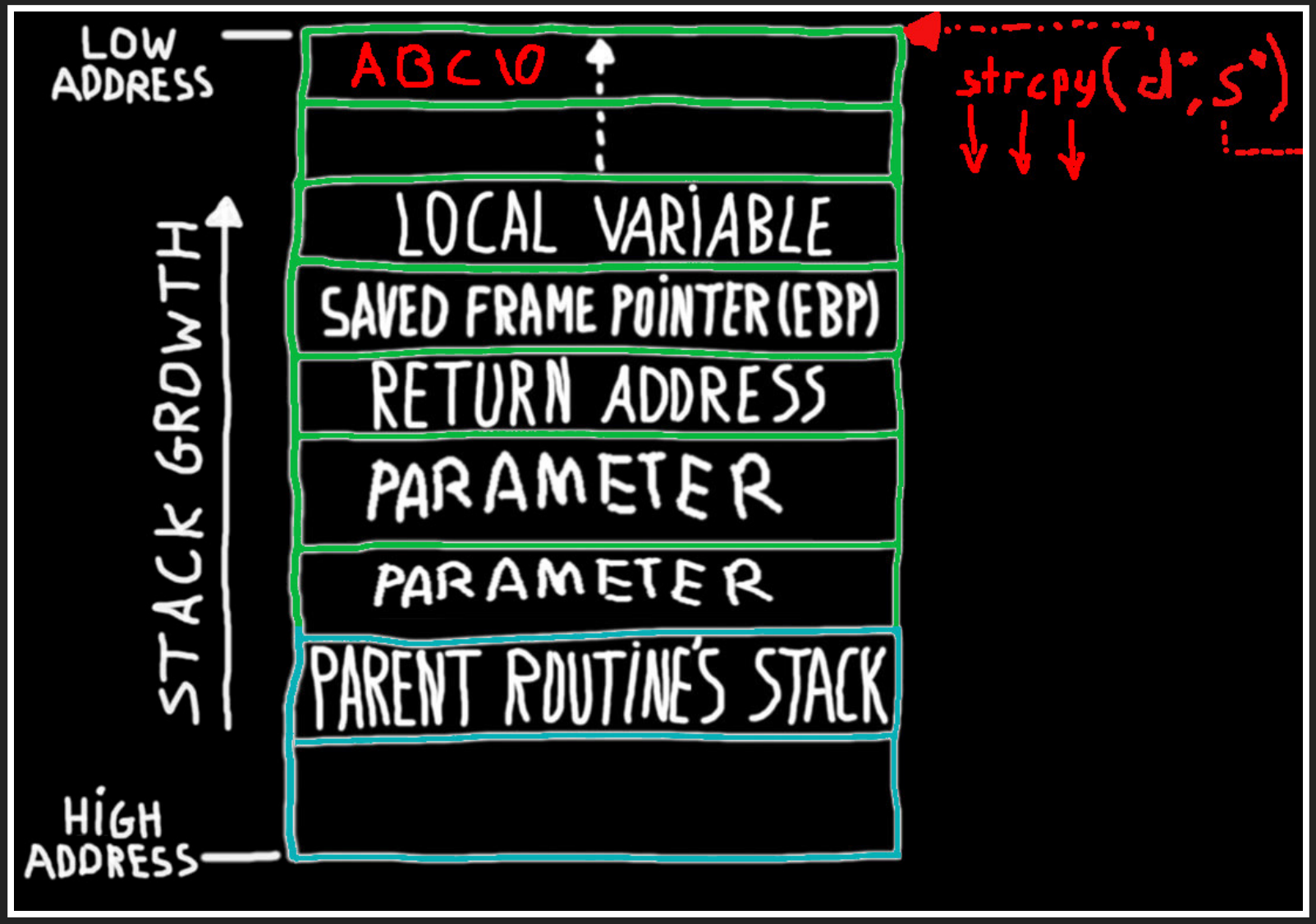
Nun soll mit `strcpy` das erste Programm Argument in den Buffer geschrieben werden.

```
char* strcpy(char *dest, char *src);
```

Kopiert solange bis ein null-byte kopiert wurde (String-Terminator)

LOKALE VARIABLEN

Platz für lokale Variable geschaffen.



Strcpy schreibt entgegen der Stack growth...

...solange bis ein NullTerminator kopiert wurde.

Enthält `*src` mehr bytes als `*dest` reserviert hat werden Bereiche überschrieben

Es gibt keine built-in safe-guards die sicherstellen, dass der reservierte Platz ausreicht

DEMO

Beliebieges Ausführen von Memory (Kontrolle EIP)

Beliebieges Schreiben und Lesen im Memory (StrCpy)

POI - EIP

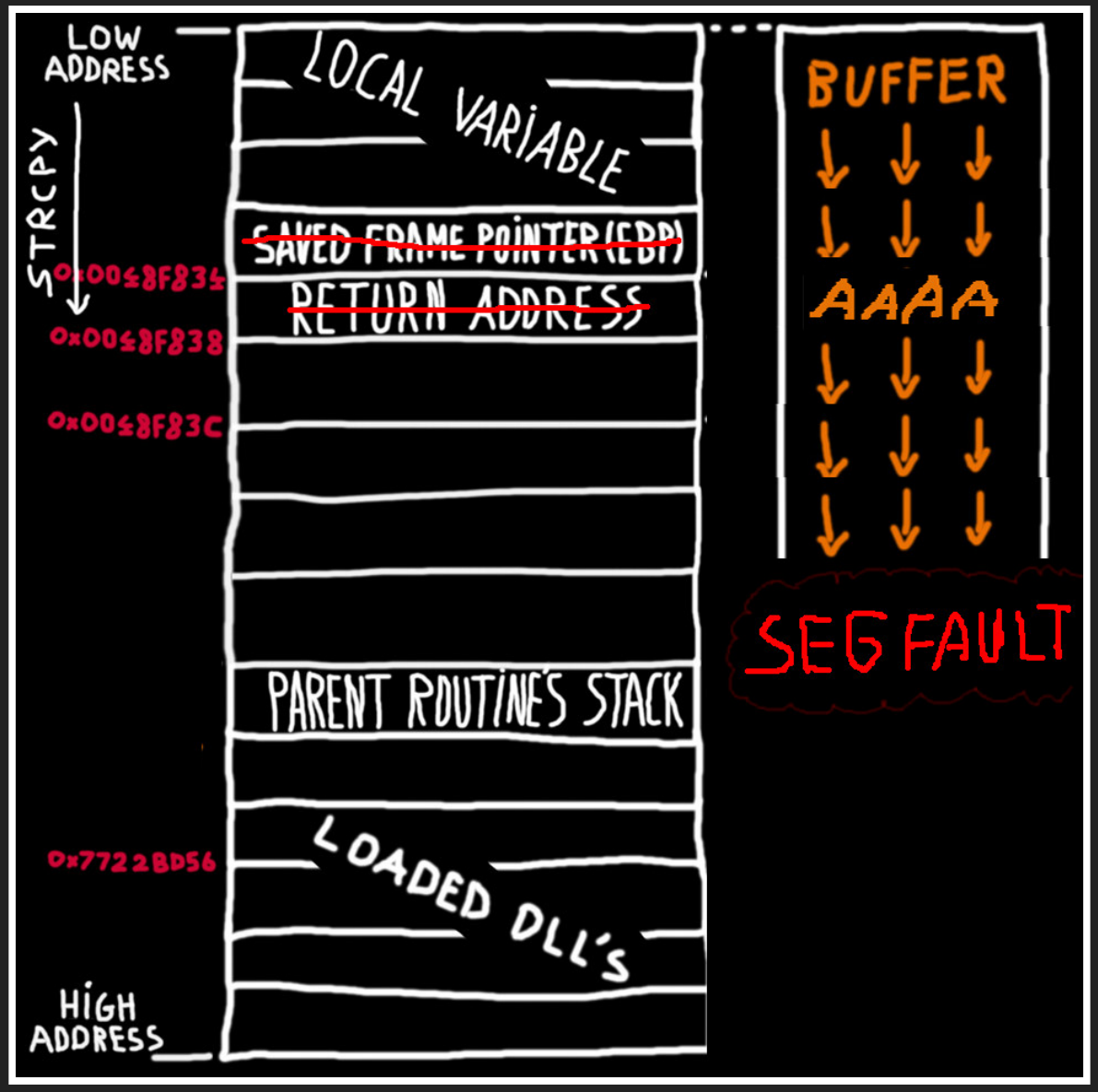
- EIP ist der Zeiger auf den auszuführenden Code
- Nachdem eine Funktion verlassen wird, wird der Return EIP von dem Stack gelesen...
- ...und in das EIP Register geschrieben.
- CPU Cycle führt den OpCode der sich an der Memory Addresse, worauf der EIP zeigt, aus
- Der Return EIP steht unterhalb der lokalen Variablen im Stackframe

POI - STRCPY

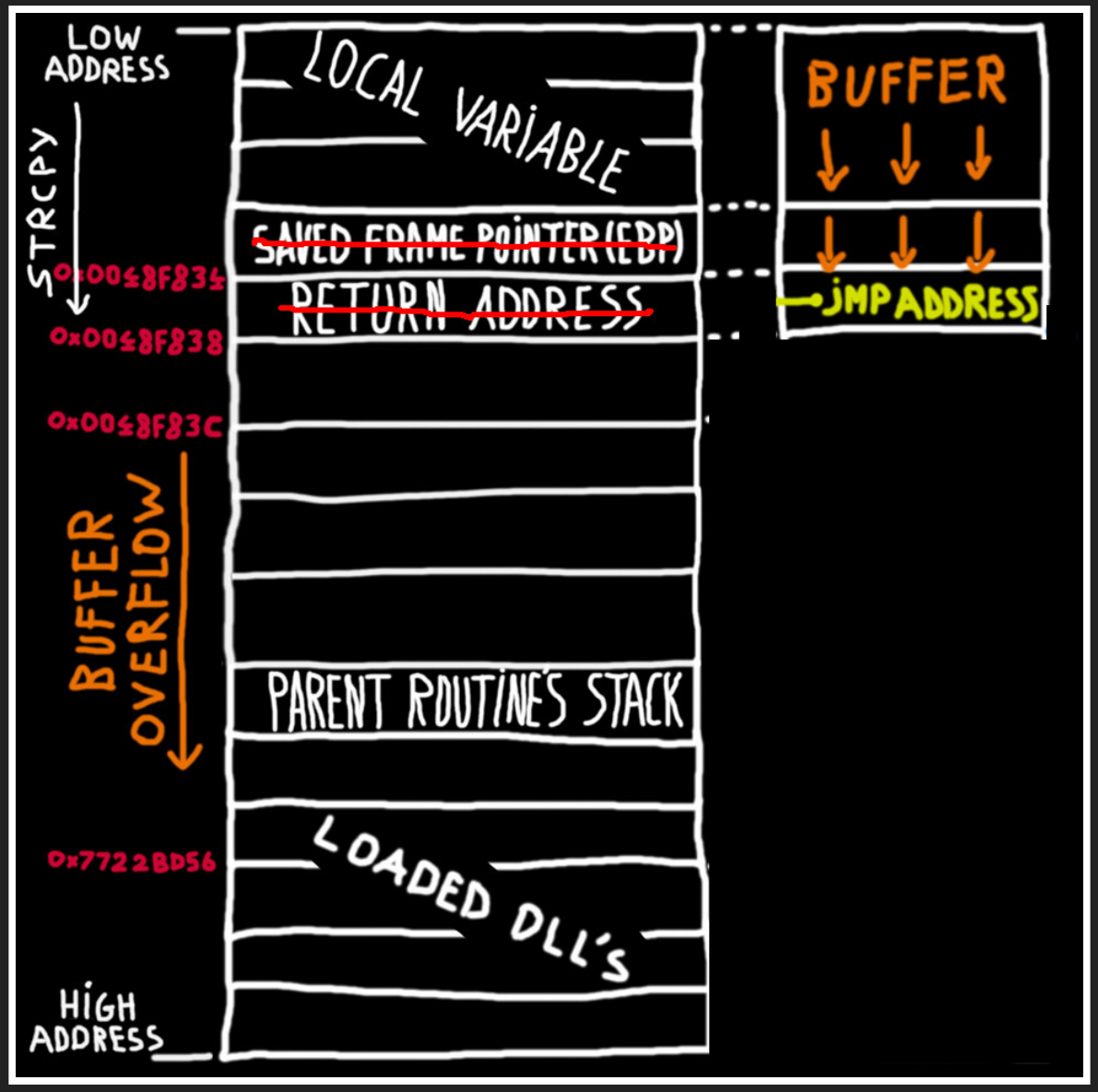
- `strcpy` schreibt entgegen der Stack growth
- und unabhängig von der Größe des Zielbuffers bis ein Null-byte kopiert wurde

BUFFEROVERFLOW

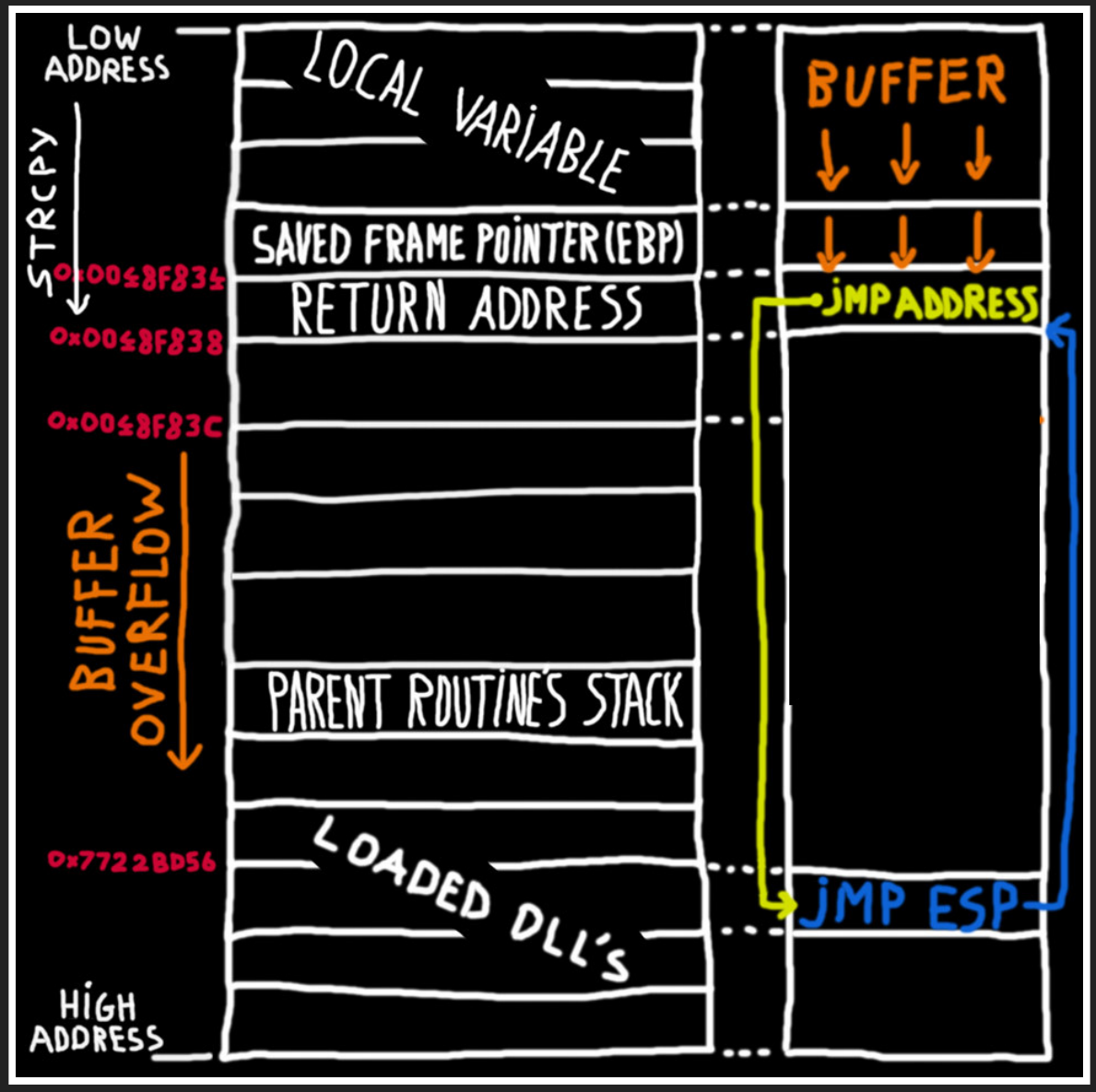
BUFFEROVERFLOW



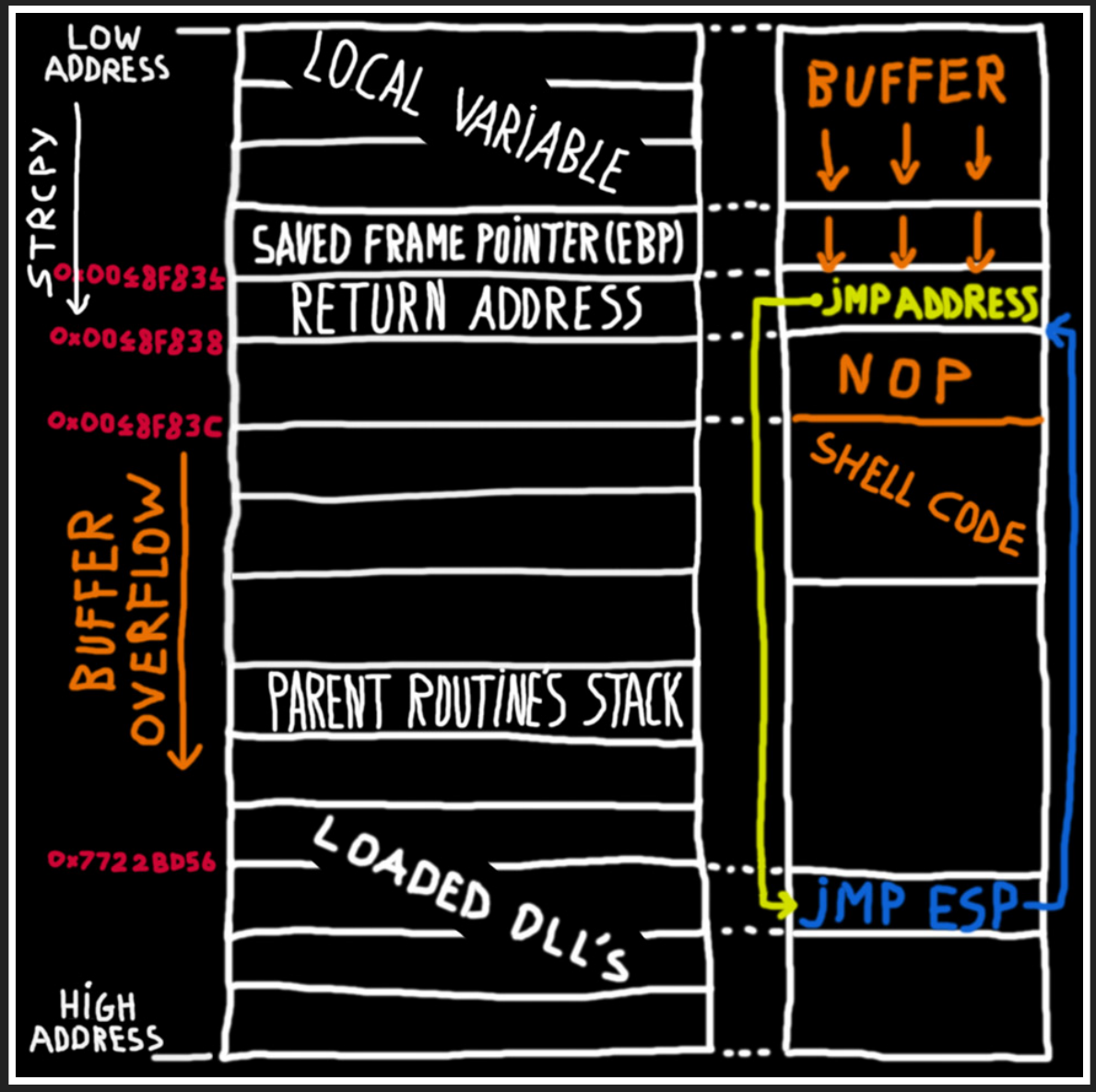
BUFFEROVERFLOW



BUFFEROVERFLOW



BUFFEROVERFLOW



TODO 4 EXPLOIT

- Pattern erstellen und EIP Offset finden
- BadChars identifizieren
- Addr für EIP finden (Shellcode ausführen z.B. Jmp ESP)
- Shellcode generieren (ggf. mit Encoder wg. BadChars)

PREREQUISITES LINUX

...for 32bit on 64bit

```
apt-get install g++-multilib libc6-dev-i386
```

...GDB Extension

Install PEDA

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

int countInputLength(char *input){
        char buffer[50];
        strcpy(buffer, input);
        int len = strlen(buffer);
        return len;
}

int main(int argCount, char *args[]){
        if(argCount < 2){
                return 0;
        }

        char input[200];
        strncpy(input, args[1], 200);

        int len = countInputLength(input);
        printf("Your input is %d characters long!\n", len);
        return 0;
}
```

COMPILE

```
gcc vulnByDesign.c -m32 -g -fno-builtin -fno-stack-protector
-z execstack -no-pie -o vulnByDesign
```

-m32	32 Bit
-g	debug information
-fno-builtin	Do not optimize library Functions never copied to the code
-fno-stack-protector	Disable CANARY
-z execstack	Disable DEP (data execution prevention)
-no-pie	Disable PIE (Position Independent Executable)

DISABLE ASLR

```
echo 0 > /proc/sys/kernel/randomize_va_space
```

oder

```
gdb-peda$ aslr [on|off]
```

START GDB

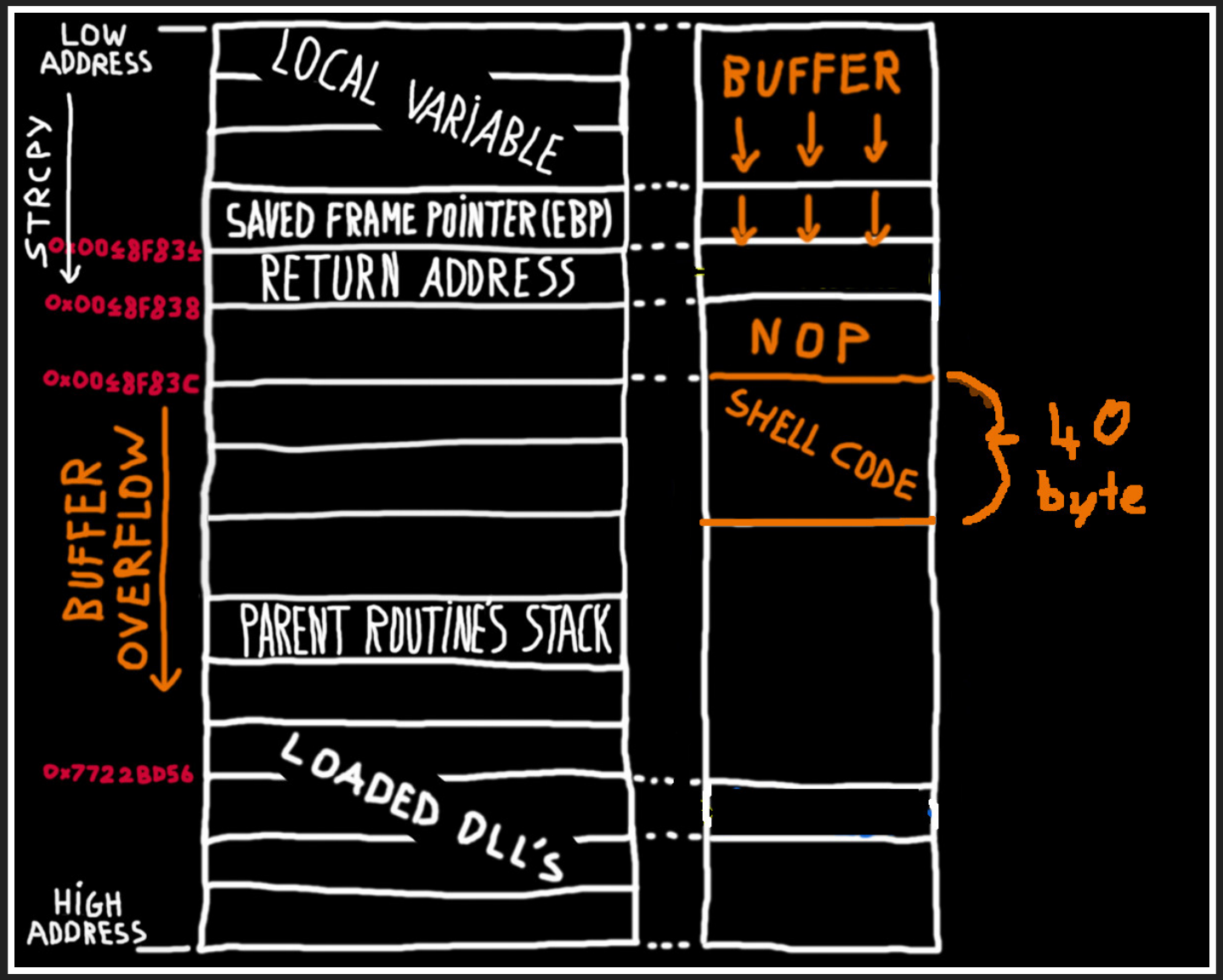
```
gdb --args ./vulnByDesign abcdef

gdb --args ./vulnByDesign `for i in {1..200}; do echo -n a;done`

gdb --args ./vulnByDesign `python -c 'print 200 * 'A'`
```


HANDS ON
WARMUP

EGGHUNTING

WHY EGGHUNTING?

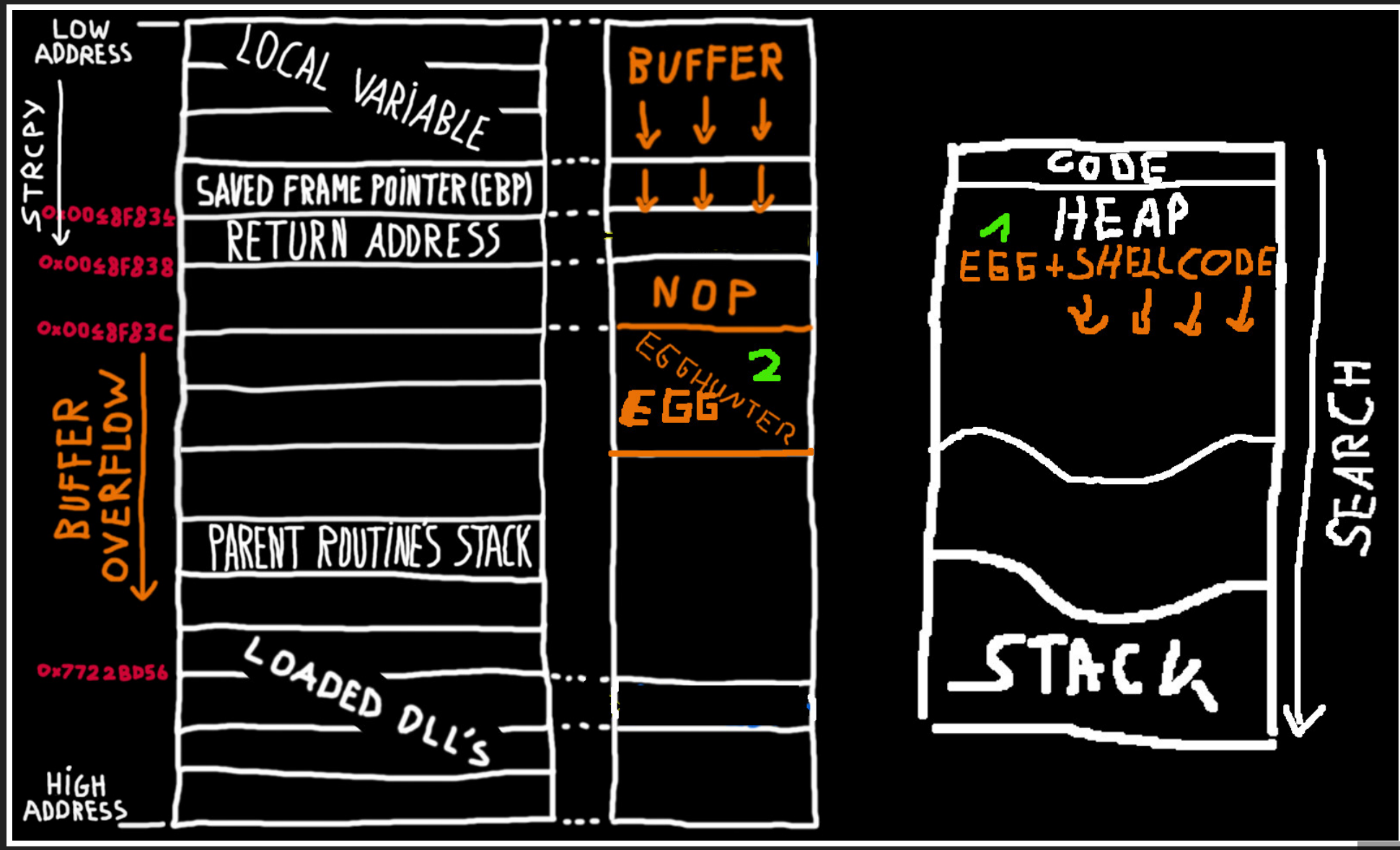
Zu wenig Platz auf dem Stack für den Shellcode - Use staged Shellcode



FUNKTIONSWEISE

- Shellcode auf den Heap/Stack
- Shellcode ist mit einem Egg markiert
- Egghunter über BufferOverflow auf den Stack
- Egghunter durchsucht den RAM nach dem Egg
- Mit Egg markierten Shellcode ausführen

FUNKTIONSWEISE



INPUTVECTORS

- Zusätzliche Eingabemöglichkeiten finden (Manuell bzw. Fuzzing)
- Nicht alle Eingaben sind dauerhaft im RAM zu finden (free()/methode wird verlassen/anderer thread/etc.)
- Eingaben müssen zum Zeitpunkt des BufferOverflows zur Verfügung stehen

PSEUDOEXPLOIT

```
egg = "{unique 4byte hex}";

shellcode = "{opens a reverse shell}";

buffer4Heap = egg + egg + shellcode;

egghunter = "{searches for eggs in RAM and jmp to it}";

buffer4Stack = offset + eip + nopsled + egghunter;
```

EGG

- unique
- eight byte egg when doing the searching

  stored once in the searching code and accidentaly run into the egg hunter itself vice running into the expected buffer
- 0x50905090 repeated twice

  (opcode for nop; push eax)
- some egg hunter implementations require that the egg itself be executable assembly

EGGHUNTER - ANFORDERUNGEN

- Robust (inaccessible mem)
- Small
- Fast

EGGHUNTER - IMPLEMENTIERUNG

- most obvious approach would be to register a **SIGSEGV handler** to catch invalid memory address
  (but egghunter size is too large)
- system call for checking if memory address is valid

Safely Searching Process Virtual Address Space


HANDS ON
EGGHUNTING - LINUX

EGGHUNTER BUGFIX

Der 35byte große Egghunter terminiert mit einem SIGSEGV.

SMALL EGGHUNTER

- Mit 18 byte sehr klein. Aber nicht robust!
- Wurde "optimiert" indem der AccessCheck entfernt wurde
- Außerdem können 4byte eggs im shellcode verwendet werdene

  Um nicht versehentlich im egghunter zu laden wird das (egg -1) als Hex gespeichert und dann im egghunter vor der Suche hochgezählt


SCRAMBLED
EGG(HUNTER)

HOW TO IDENTIFY VULNERABILITIES AND INPUTVECTORS

- Manual testing
- Code Review
- Static Code Analysis
- Fuzzing
- ...

FUZZING

FUZZING

Fuzzing is a process of
sending arbitrary/malicious/random data
to a program in order to
generate failures, or errors in the application

FUZZING

typically **local** analysis technique and not something we would typically use in a remote penetration test.

FUZZING

- Identify Inputs
- Generate Fuzzed Data (Define Data Schema)
- Send Fuzzed Data
- Monitor for Exception, Log and Restart
- Determine Exploitability

IDENTIFY INPUTS

Prior to engaging in a fuzzing test

- It is important to understand the protocol you will be evaluating
- Identify all input that the application accepts and processes

MONITOR FOR EXCEPTION, LOG AND RESTART

we typically need to attach a debugger to the target process to capture stack and register dump information.

CHOOSE TOOL

- Manually
- Code yourself (Python,...)
- Fuzzer-Tool with config (sFuzz, Peach, etc.)
- Frameworks (sully, kitty, SPIKE, etc.)

Config vs. Coding

fuzzing resources

SIMPLE FUZZER

Fuzzing mit sfuzz

```
sfuzz -S 127.0.0.1 -p 9999 -f fuzz.vulnserver.config -TOev
```

SIMPLE FUZZER CONFIG

```
#generates sequences incremented by and max
seqstep=1000
maxseqlen=10000
literal=abcdefg
sequence=A
sequence=1
#Overwrite Line Terminator
!CRLF=0d 0a
lineterm=CRLF
# wait between requests (default 100ms)
reqwait=200
endcfg
STATS FUZZ

--
GMON FUZZ

--
```

BOOFUZZ

Fork from Sully

Install boofuzz

FUZZING FOR BUFFEROVERFLOW

- Bufferlänge verändern
- Bedingungen (IF) im Code berücksichtigen
  (Zeichen, Länge, Reihenfolge der Aufrufe etc.)
- Ist die Länge ein Parameter?

FUZZING FOR EGGSHELLCODE SPACE

- Bufferlänge verändern
- Bedingungen (IF) im Code berücksichtigen
  (Zeichen, Länge, Reihenfolge der Aufrufe etc.)
- Nicht alle Eingaben sind dauerhaft im RAM zu finden
  (free()/methode wird verlassen/anderer thread/etc.)

FUZZING FOR EGGSHELLCODE SPACE

```
sendPatternData()

sendBufferOverflowExploit()

memDump = DumpMemory()

searchForPatternData(memDump)
```

PREPARE 4 WINDOWS

- Lunch vulnserver.exe on Desktop
- Start Immunity Debugger with Mona
- File > Attach

- Or use debugVulnserver.bat

PREPARE 4 WINDOWS

- !mona findmsp
- !mona jmp -r {register}
- !mona pattern_create
- !mona pattern_offset
- !mona asm -s {assembler instruction}

HANDS ON

FUZZING & EGGHUNTING - WINDOWS