Computersystemsicherheit

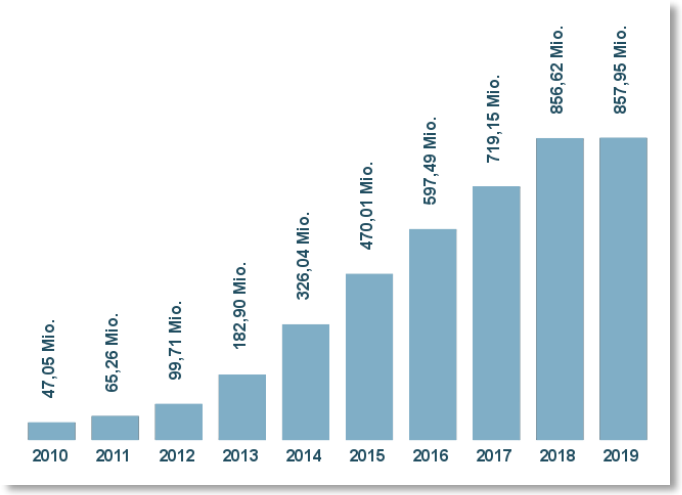
Prof. Marc Fischlin, Wintersemester 18/19

06

Betriebssystem-Sicherheit

Malware

Anzahl bekannter Malware
2007-2018 (ca. 850.000.000)
laut AV-TEST Institut



TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Arten von Malware = Malicious Software

Wurm

verbreitet sich selbst
über Netzwerk

beeinträchtigt
„nur“ Leistung

Virus

böswillige Software,
die sich selbst verbreitet

hängt sich an andere
Software/Systeme an

verändert Aussehen
(„Polymorphie“)

Trojaner

Software mit zusätzlicher,
böswilliger Funktion

repliziert sich
nicht von selbst

oft auch: Trojaner-Virus,
wenn nicht-selbst-replizierender Virus
zusätzliche Funktionen wie
Passwort-Sniffing auf System installiert

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Spezialfälle

Ransomware – Torjaner-Virus, der Dateien verschlüsselt und nur gegen „Lösegeld“-Zahlung wieder freigibt (oder auch nicht)

Scareware – Trojaner, der als Virenscanner vermeintlich gefundene Viren gegen Bezahlung löschen soll

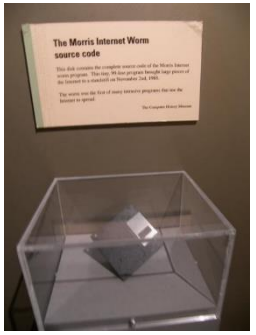
Zero-Day-Exploits:

neue Schwachstellen,
bei der Entwickler keine Zeit hat (zero days),
um Security Patch bereitzustellen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Bekannte Malware

Quelle: Wikipedia.de



1982 Elk Cloner	erster Virus (für Apple-Computer)
1988 Morris-Wurm	legte große Teile des Internets lahm
2000 Loveletter	Wurm/Virus, per E-Mail mit Betreff: ILOVEYOU verteilt überlastete Mail-Server, löschte auch Dateien 50 Millionen Computer, ca. 10 Milliarden US-$ Schaden
2003 Slammer	Wurm, infizierte in 15min große Teile des Internets geschätzter Schaden ca. 1 Milliarde US-$
2010 Stuxnet	befiel Industrie-Anlagen, vor allem im Iran
2013 CrytoLocker	Ransomware, per E-Mail-Anhang verteilt mind. 500.000 Computer infiziert, 30 Mio. US-$ Schaden
2017 WannaCry	Ransomware, verbreitet über alte Windows-Versionen mind. 220.000 Rechner infiziert, nur 30K US-$ Lösegeld

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Malware verbreiten

Gegenmaßnahmen:
Antivirenprogramm;
nur vertrauenswürdige Quellen;
System aktualisieren

klassisch – als E-Mail-Anhang
oder während einer Programm-Installation
oder per Netzwerkprotokollem

drive-by-download –
beim Besuch einer Web-Seite
(via Java, Javascript, Flash,…)

Gegenmaßnahmen:
Antivirenprogramm;
NoScript;
Sandboxing

physisch – durch infizierte Tokens

Gegenmaßnahmen:
Antivirenprogramm;
nur vertrauenswürdige Quellen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Malware per Tokens verbreiten

S&P 2016

angeblich auch Stuxnet per USB-Token

Users Really Do Plug in USB Drives They Find

Matthew Tischer† Zakir Durumeric‡† Sam Foster† Sunny Duan†
Alec Mori† Elie Bursztein◇ Michael Bailey†

† University of Illinois, Urbana Champaign ‡ University of Michigan ◇ Google, Inc.

{tischer1, sfoster3, syduan2, ajmori2, mdbailey}@illinois.edu
zakir@umich.edu elieb@google.com

Abstract—**We investigate the anecdotal belief that end users will pick up and plug in USB flash drives they find by completing a controlled experiment in which we drop 297 flash drives on**

median time to connection of 6.9 hours and the first connection occurring within six minutes from when the drive was dropped. Contrary to popular belief, the appearance of a drive does not

ließen ca. 300 Flash-Drives mit „gutartiger Malware“ auf US-Campus liegen

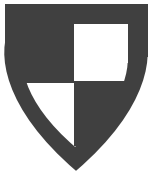
„Malware“ pingte Heimatrechner an, wenn bestimmtes File geöffnet

45% der verteilten Sticks meldeten sich

wurde vorher durch Institutional Review Board (IRB) der Universität abgesegnet

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity Cryptography & Complexity Theory Technische Universität Darmstadt www.cryptoplexity.de

Methoden der Antivirenprogramme



Systemscan – sucht auf Festplatte nach verdächtigen Dateien

Real-Time Detection (On-Access Scanning) –

überprüft beim Speicher, Öffnen, Ausführen, …

reaktiv

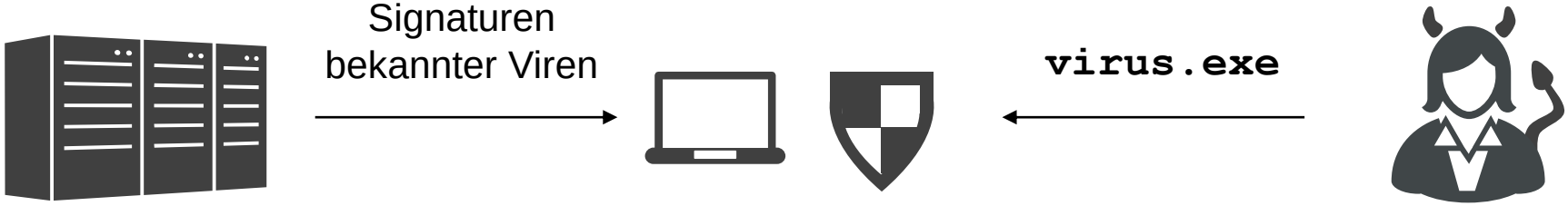
proaktiv

Activity Monitoring –

prüft auf virentypisches Verhalten,
z.B. Replikation in AutoStart, DNS-Anfragen,…

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Reaktives Scannen



AV-Software-Hersteller

prüfe, ob Signatur bekannt

prüfe auch auf Mutationen,
z.B. wenn Virus verschlüsselt,
dann sich erst in Sandbox entschlüsseln lassen

simuliere
`virus.exe`

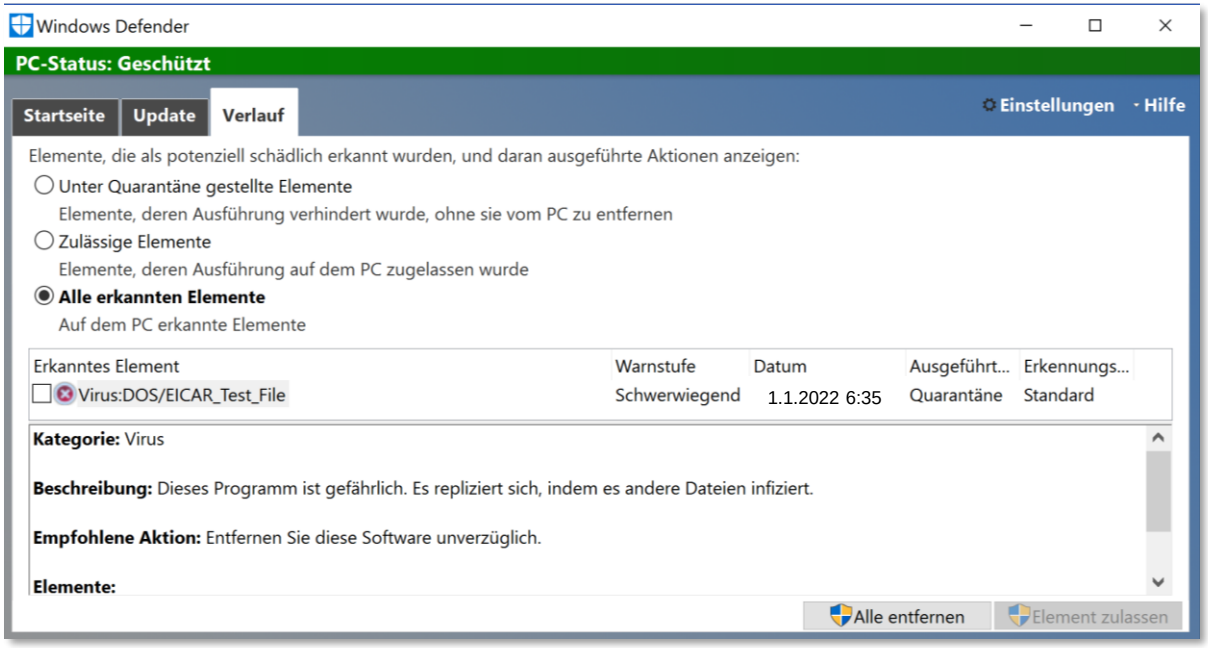
Sandbox

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

EICAR-Test für Virenschutz

einfaches Testen von Antiviren-Programmen durch (harmlose) Testdatei von EICAR (European Institute for Computer Anti-Virus Research e.V.):

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```



TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Effektivität von Virenscannern?

"People thought that virus protection protected them,
but we can never block all viruses."

Trend Micro CEO Eva Chen in einem ZDNet-Interview von 2008

Erfolgsstatistik von AV-Software variiert zwischen 40% und 99.9%

Tests beziehen sich aber meistens auf bekannte Massen-Malware

Erfolgsquote bei neuen gezielten Virenangriffen unklar

Was ist mit Verzögerung bei Signatur-Updates?

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

(Stack) Buffer Overflows

Aufspielen von Malware-Dateien $\rightarrow$ Angriff auf Ausführung „gutartiger“ Software

Heap-Overflows für dynamisch allokierten Speicher funktionieren analog

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

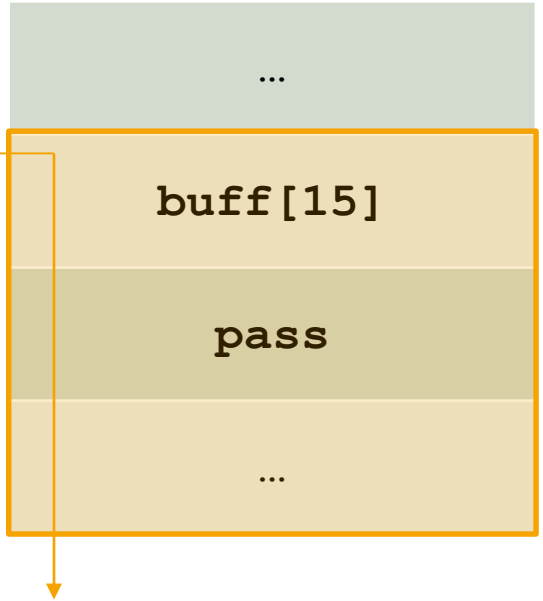
Buffer Overflow: Prinzip

Funktionsaufruf schreibt
lokale Variablen auf den Stack

```
int main(void)
{
     char buff[15];
     int pass = 0;

     gets(buff);
     …

     return 0;
}
```

C-Programm

Stack



Speicheradresse

0xffffcba0

0xffffcbbc

Funktion prüft nicht,
ob Länge der
Eingabe beschränkt

**gib längere Eingabe und
überschreibe weitere Teile des Stacks,
z.B. Variable `pass` oder Systemvariablen**

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt

Einfaches Beispiel

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

int main(void)
{
    char buff[15];
    int pass = 0;

    printf("\n Enter password: \n");

    gets(buff);

    if(strcmp(buff, "123456"))
    {
        printf ("\n Wrong Password \n");
    }
    else
    {
        printf ("\n Correct Password \n");
        pass = 1;
    }
    if(pass)
    {
       /* give root or admin rights to user*/
        printf ("\n Root privileges given to user \n");
    }
    return 0;
}
```

zu lange Passworteingabe:

AAAAAAAAAAAAAA...AAAAAA

schreibt Wert ≠ 0 in **pass**

und gibt dann Programm Administrator-Rechte, sogar für falsches Passwort

nach:
Himanshu Arora
www.thegeekstuff.com/2013/06/buffer-overflow/

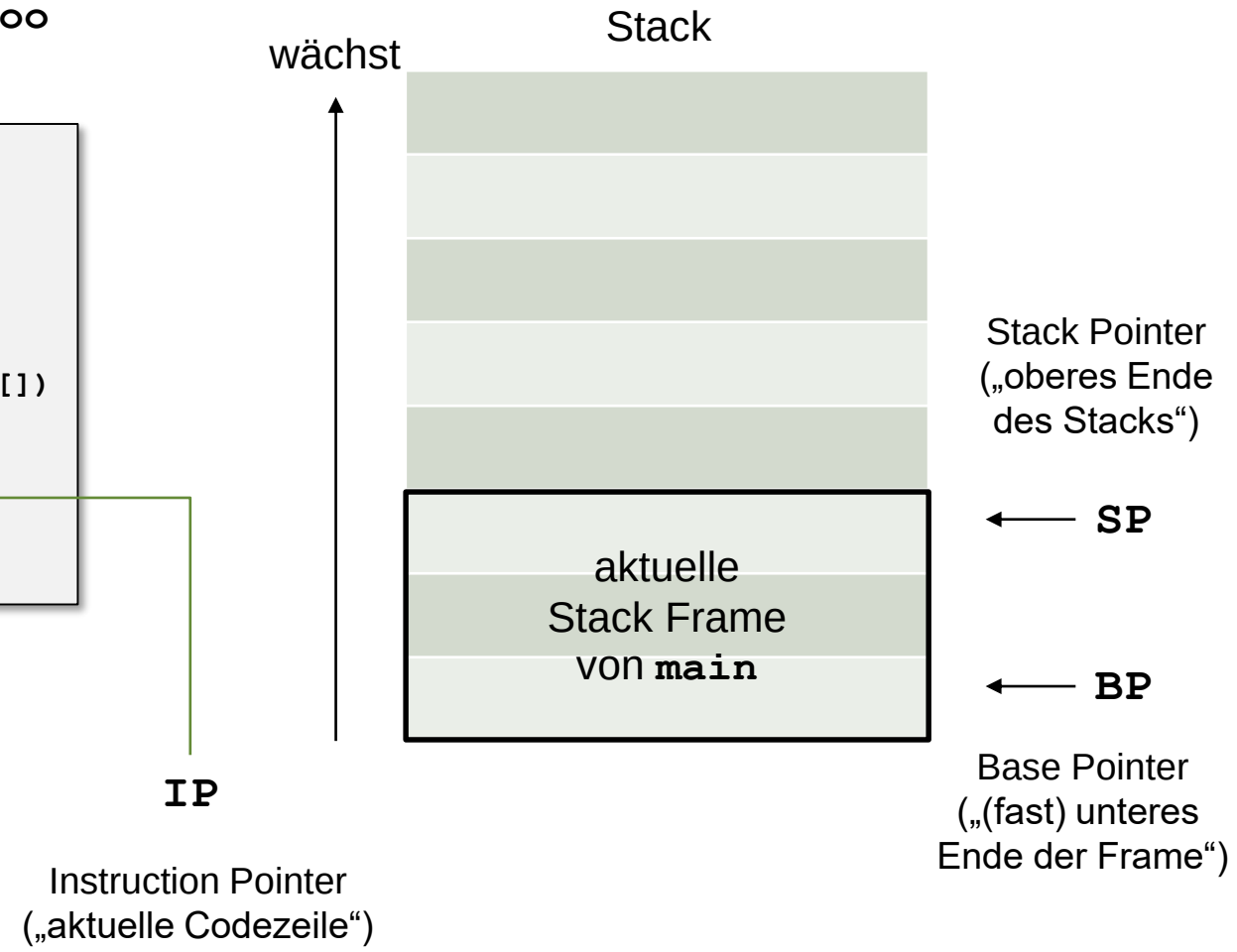
TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt

Smashing the Stack: Grundlagen

Mudge: How to Write Buffer Overflows, 1995
Aleph One: Smashing the Stack for Fun and Profit, 1996

vor Ausführung des Aufrufs **`foo`**

```
void foo(char* input) {
   char buf[10];
   …
}


int main(int argc, char* argv[])
{
    …
    foo(argv[1]);
    …
}
```

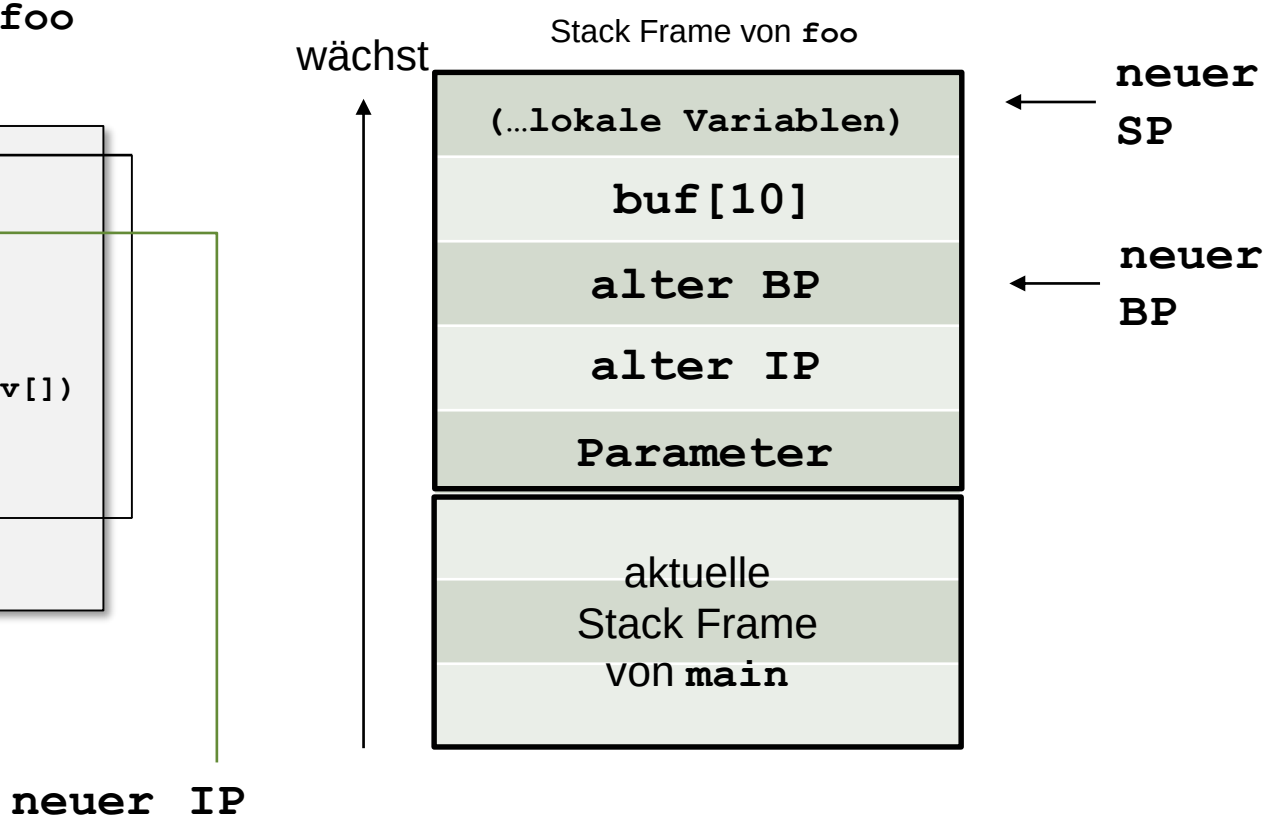


TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Smashing the Stack: Grundlagen

Ausführung des Aufrufs **foo** (Prolog)

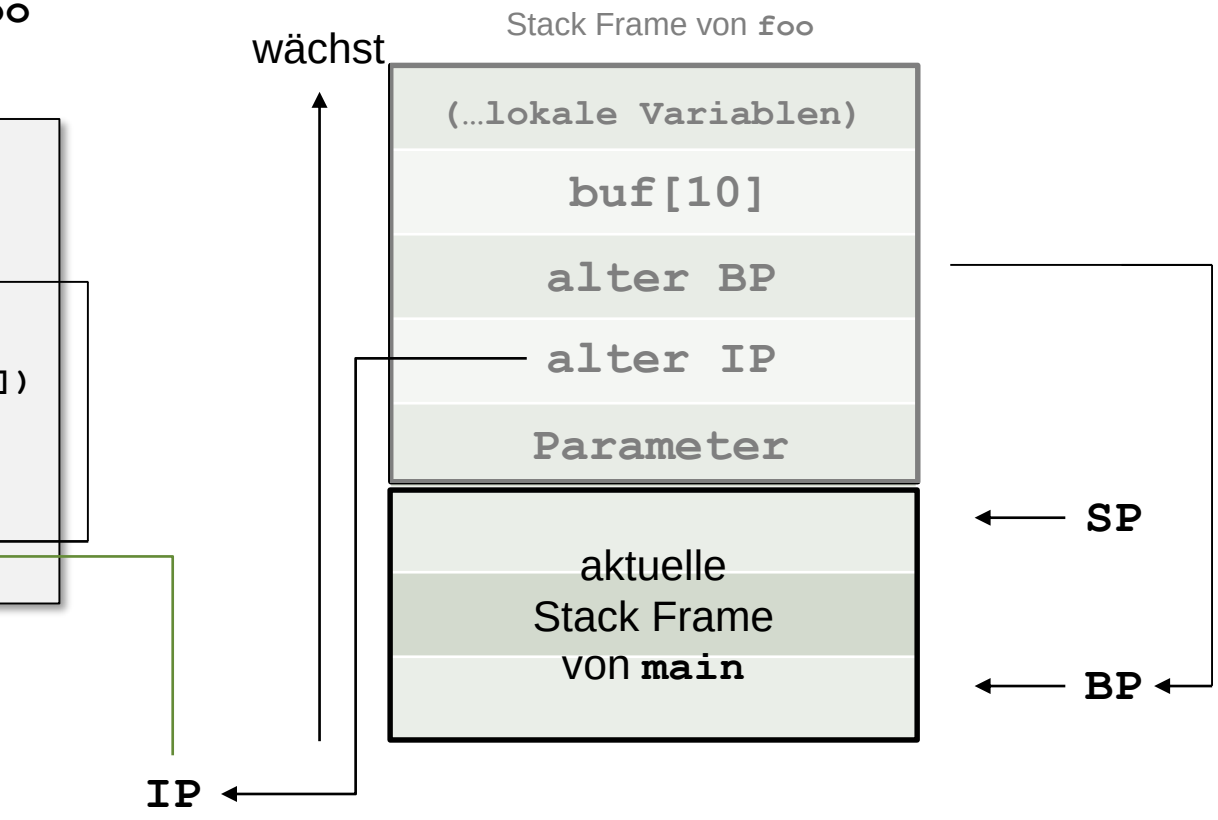
```
void foo(char* input) {
   char buf[10];
   …
}


int main(int argc, char* argv[])
{
    …
    foo(argv[1]);
    …
}
```



Smashing the Stack: Grundlagen

Ausführung des Aufrufs **`foo`**
(Epilog)

```
void foo(char* input) {
   char buf[10];
   …
}

int main(int argc, char* argv[])
{
    …
    foo(argv[1]);
    …
}
```



Smashing the Stack: Beispiel

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

void foo(const char* input)
{
    char buf[10];

    strcpy(buf, input);
}

void bar(void)
{
    printf("Wrong code being executed\n");
}

int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        printf("Please supply a string as an argument!\n");
        return -1;
    }

    foo(argv[1]);
    return 0;
}
```

Angriffsidee:

Überschreibe `alter IP` auf Stack, so dass Programm nach Epilog von `foo` in `bar` fortgesetzt wird

nach:
Thomas Schwarz
COEN 152 Forensics, Buffer Overflow Attack

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Ausführung mit Debugger **gdb** (Prolog von **foo**):

```
(gdb) info frame
Stack level 0, frame at 0xffffcb90:
 rip = 0x1004010ec in foo (boverflow2.c:12); saved rip = 0x10040119c
 called by frame at 0xffffcbd0
 source language c.
 Arglist at 0xffffcb80, args: input=0xffffcc60 …
 Locals at 0xffffcb80, Previous frame's sp is 0xffffcb90
 Saved registers:
  rbp at 0xffffcb80, rip at 0xffffcb88, xmm15 at 0xffffcb88
(gdb) x/20x $sp
0xffffcb50:     0x235d7b84      0x000006fe      0x801fc870      0x00000001
0xffffcb60:     0x00403074      0x00000001      0xffffcba0      0x00000000
0xffffcb70:     0x8021af56      0x00000001      0xffffcb98      0x00000000
0xffffcb80:     0xffffcbc0      0x00000000      0x0040119c      0x00000001
0xffffcb90:     0xffffcc60      0x00000000      0x0040111e      0x00000001
(gdb) print &buf
$1 = (char (*)[10]) 0xffffcb70
```

Informationen Stack frame von **foo**

Inhalt des Stacks von **foo**

Adresse der lokalen Variable **buf**

alter BP **buf[10]** **alter IP**

hier soll die Adresse unseres Ziels `bar = 0x100401103` stehen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity Cryptography & Complexity Theory Technische Universität Darmstadt

Ausführung mit Debugger **gdb** (**Epilog** von **foo**):

```
(gdb) info frame
Stack level 0, frame at 0xffffcb90:
 rip = 0x1004010ec in foo (boverflow2.c:12); saved rip = 0x10040119c
 called by frame at 0xffffcbd0
 source language c.
 Arglist at 0xffffcb80, args: input=0xffffcc60 …
 Locals at 0xffffcb80, Previous frame's sp is 0xffffcb90
 Saved registers:
  rbp at 0xffffcb80, rip at 0xffffcb88, xmm15 at 0xffffcb88
(gdb) x/20x $sp
0xffffcb50:     0x235d7b84      0x000006fe      0x801fc870      0x00000001
0xffffcb60:     0x00403074      0x00000001      0xffffcba0      0x00000000
0xffffcb70:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffcb80:     0xffffcbc0      0x01010101      0x00401103      0x00000001
0xffffcb90:     0xffffcc60      0x00000000      0x0040111e      0x00000001
(gdb) print &buf
$1 = (char (*)[10]) 0xffffcb70
```

alter IP zeigt nun auf bar!!!

(und BP ist defekt, aber **bar** wird erst noch ausgeführt)

Argument wurde in **buf** kopiert:

AAAAAAAAAAAAAAAA\xc0\xcb\xff\xff\x01\x01\x01\x01\x03\x11\x40\x00

16x **A** = **\x41**

überschreibe alten BP
(Achtung: **\x00** lässt **strcpy** terminieren,
daher hier „unsinniger“ Pointer mit **\x01**)

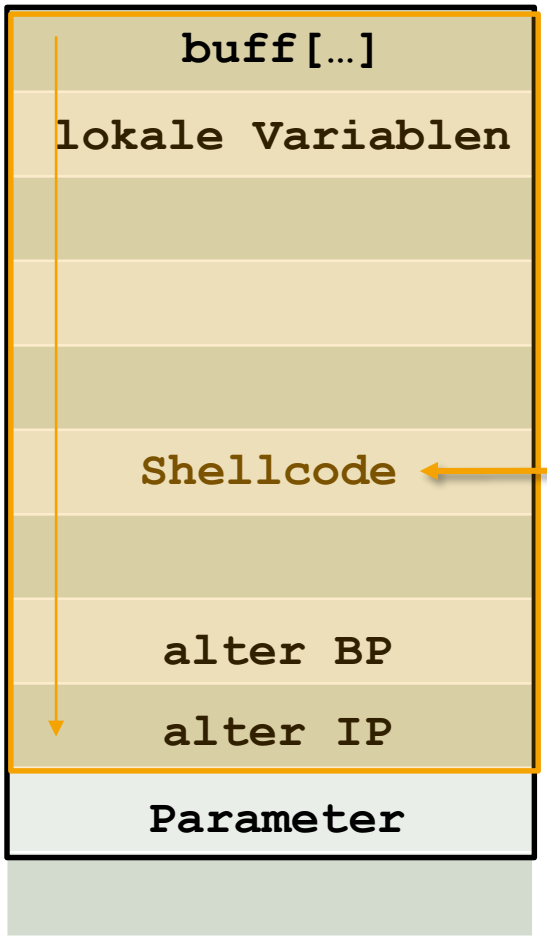
überschreibe
(Teile des) alten IP
mit **bar** Adressteil

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Eigenen Programmcode ausführen

Beispiel:
`/bin/sh` ausführen

Stack Frame der angegriffenen Routine



durch Overflow umsetzen

Shellcode einfügen

aus: Smashing the Stack for Fun and Profit

```
movl    %esp,%ebp
subl    $0x8,%esp
movl    $0x80027b8,0xfffffff8(%ebp)
movl    $0x0,0xfffffffc(%ebp)
pushl   $0x0
leal    0xfffffff8(%ebp),%eax
…
```

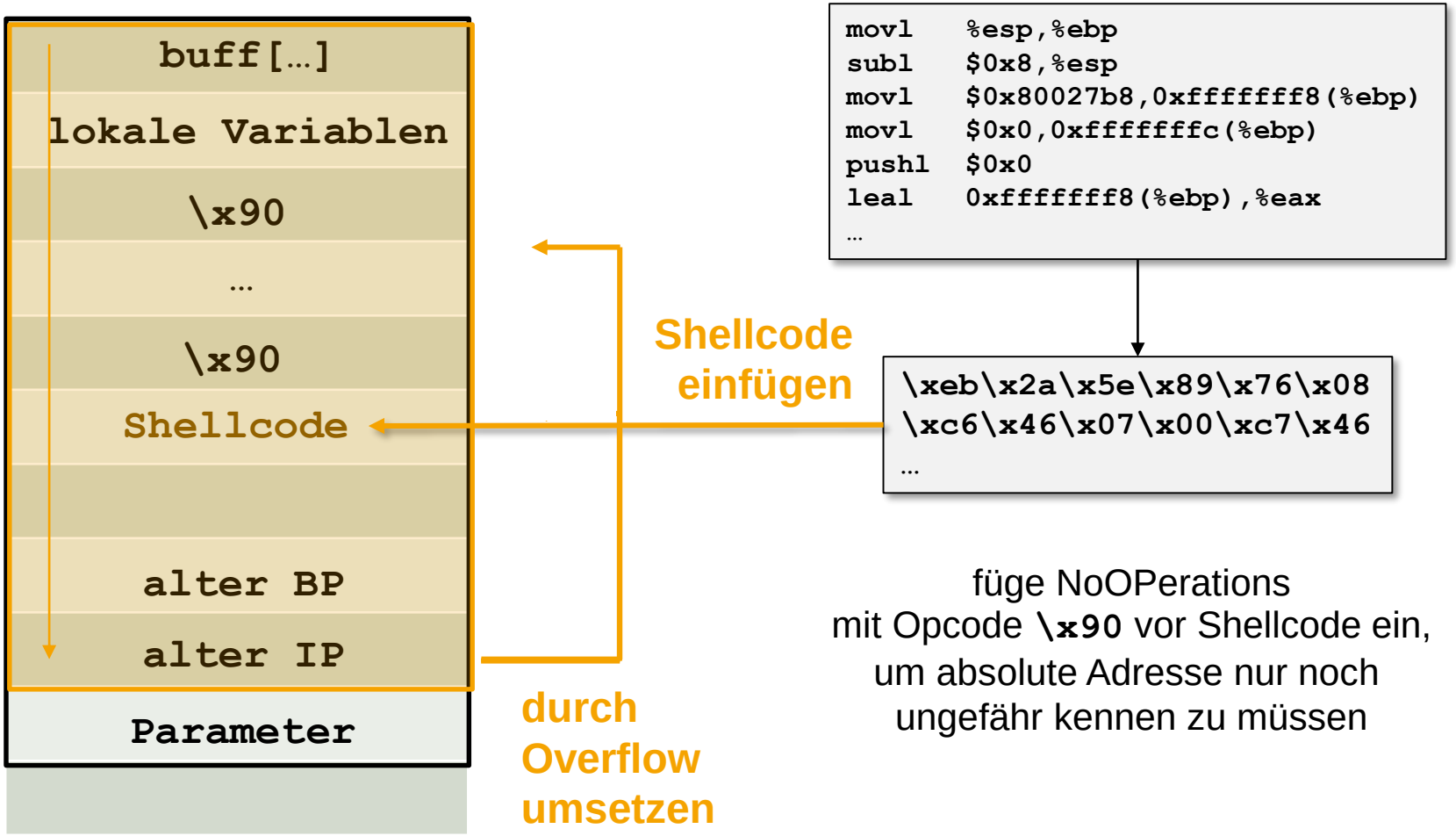
```
\xeb\x2a\x5e\x89\x76\x08
\xc6\x46\x07\x00\xc7\x46
…
```

Shellcode:
Opcode eines Assembler-Programms

`(\x00` im Shellcode evtl. durch andere Operationen ersetzen, sonst wieder `\x00`-Terminierungsproblem)

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

NOP-Slide/Sled



füge NoOPerations
mit Opcode **\x90** vor Shellcode ein,
um absolute Adresse nur noch
ungefähr kennen zu müssen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt

Gegenmaßnahmen (Programmieren)

(auch Warnungen von Compilern)

„Gefährliche“ Operationen wie **`strcpy`** vermeiden…

…ist gar nicht so einfach:

`strcpy(buf,src)` → **`strncpy(buf,src,size)`**

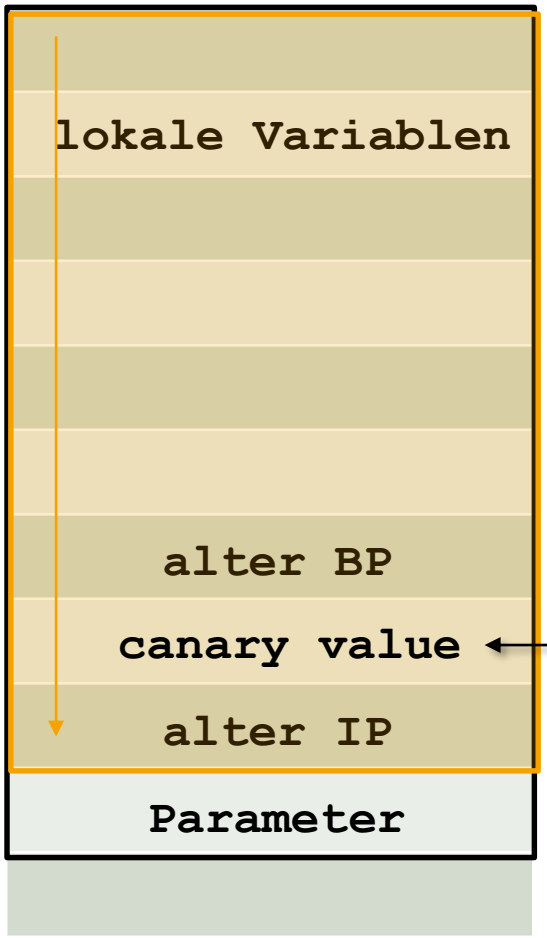
wobei maximal **`size`** so gewählt werden kann, dass nicht über **`buf`** hinaus geschrieben wird

aber immer noch Buffer Overflows möglich, da kein terminierendes Nullbyte an kopierten Wert angefügt wird und keine Prüfung auf Größenzulässigkeit

→ Varianten **`strcpy_s`** und **`strncpy_s`** kopieren Nullbyte mit und nur so viele **`char`**‘s, wie in **`buf`** passen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Gegenmaßnahmen (Stackverwaltung)



füge beim Prolog zufälligen Wert („canary“) vor **`alter IP`** ein

erzeuge und speichere **`canary value`** in globaler Variable bei Programmstart

prüfe beim Epilog, dass **`canary value`** noch korrekt

Angreifer kann **`alter IP`** nur überschreiben, wenn er **`canary value`** zerstört

(verhindert aber nicht DoS-Angriff)

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt

Gegenmaßnahmen (Systemebene)

ASLR – Address Space Layout Randomization

weist Stack, Heaps, usw. zufällige Adressen zu
→ Angreifer kann Instruction Pointer nicht mehr so leicht richtig setzen

erschwert Angriffe, aber nicht unmöglich, ASLR zu umgehen
(wenig Wahlmöglichkeiten im Adressraum,
NOP slides bzw. Verteilen des Codes an mehreren Stellen: Spraying)

DEP – Data Execution Prevention (alias W$_{rite}$ ⊕ eX_{ecute})

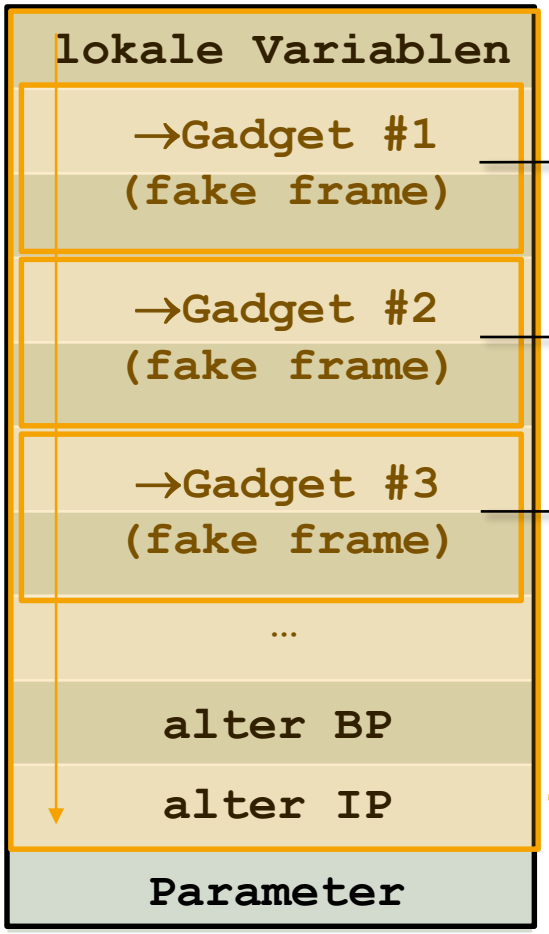
Teile des Speichers werden als nicht ausführbar markiert
→ Angreifer kann Shellcode nicht mehr ausführen lassen

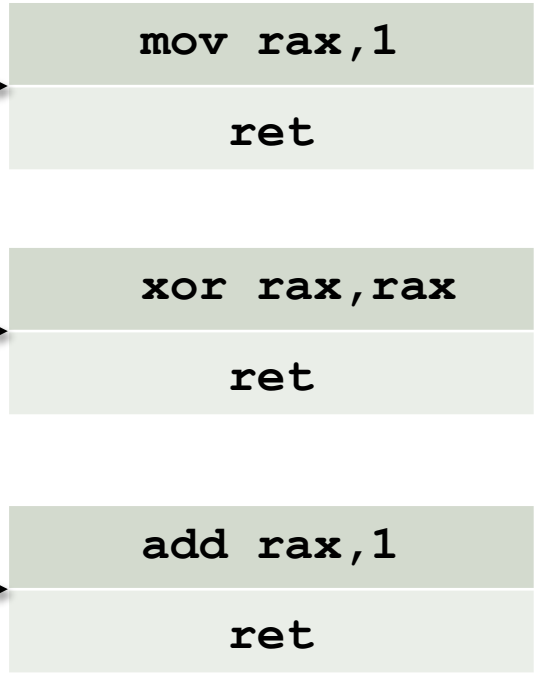
kann durch Return-Oriented Programming umgangen werden

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Idee von Return-Oriented Programming

(Teile von) Programme(n), die üblicherweise im Speicher vorhanden sind (z.B. dll's)





Stack selbst wird nicht ausgeführt (kein DEP)!

Gadget-Sequenzen für komplexe Programme lassen sich automatisch zusammenbauen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Gegenmaßnahmen gegen ROP

ASLR – Address Space Layout Randomisation

hilft auch hier (bedingt)

CFI – Control-Flow Integrity

oft umgesetzt mit „Shadow Stack“, der Integrität des Stacks garantiert

geht mit Performance-Verlusten einher

Control-flow Enforcement Technology (CET)
auf Hardware-Ebene

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Nennen Sie die Unterschiede zwischen Würmern, Viren und Trojanern.

Erklären Sie die grundlegende Idee von Buffer-Overflow-Angriffen.

Nennen Sie weitere C-Operationen wie **`strcpy`**, die Sie für unsicher halten.

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Isolation

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Confinement:

Anwendung kann keinen Schaden im Rest des Systems erzeugen

kann implementiert werden durch

Isolation:

jede Anwendung läuft in eigener Umgebung

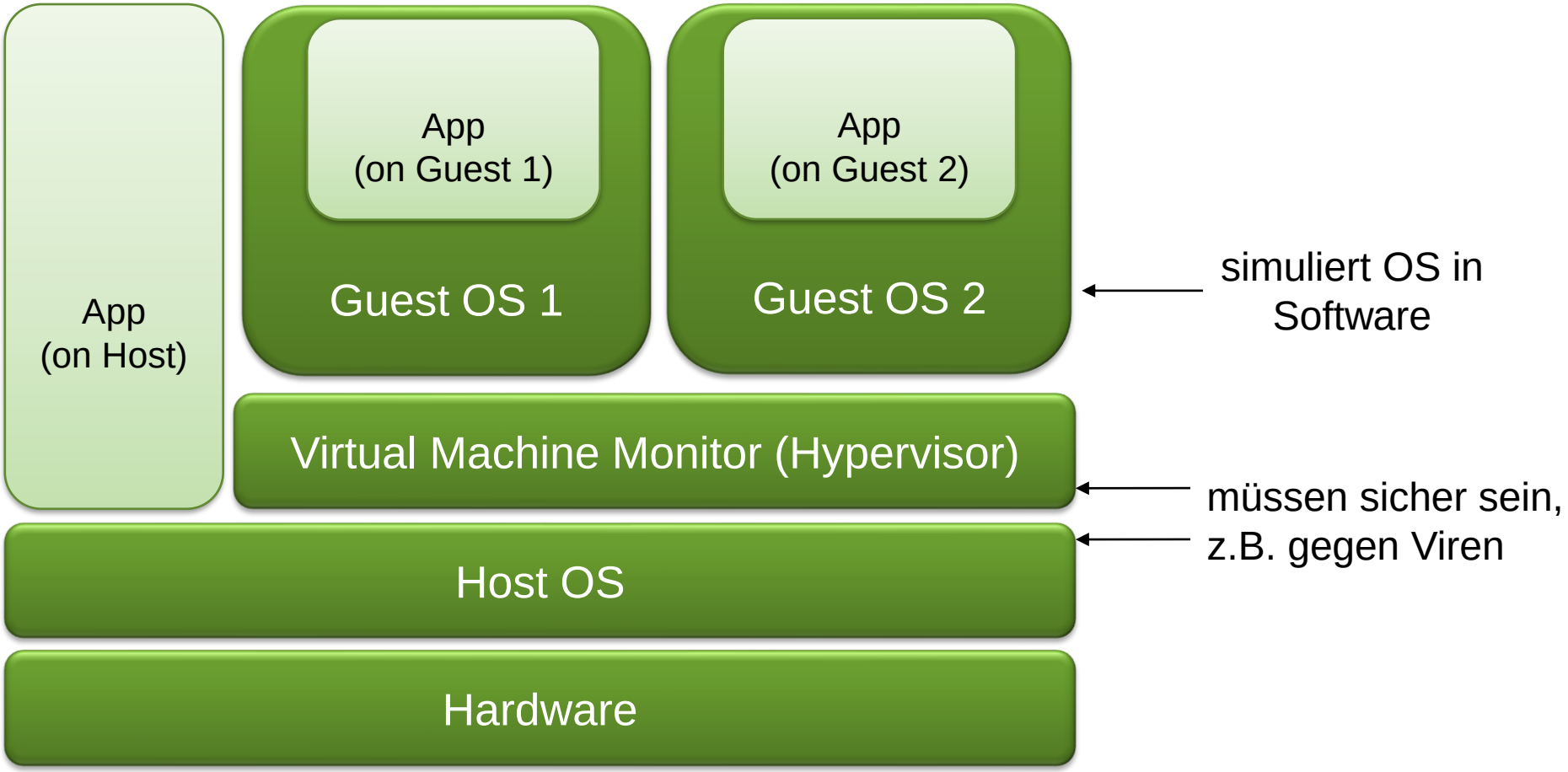
kann implementiert werden durch

Sandboxing:

„eigene“ Software-Umgebung auf gemeinsamen System

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Sandboxing durch VMs



TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Container

Container – alternatives Sandboxing-Prinzip zu VMs

mehrere Container teilen sich OS

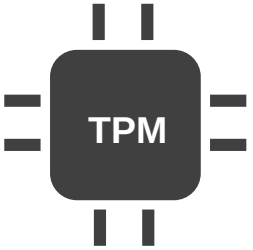
„leichtgewichtiger“ als VMs (Größe, Geschwindigkeit)

Intel Software Guard Extensions (SGX)

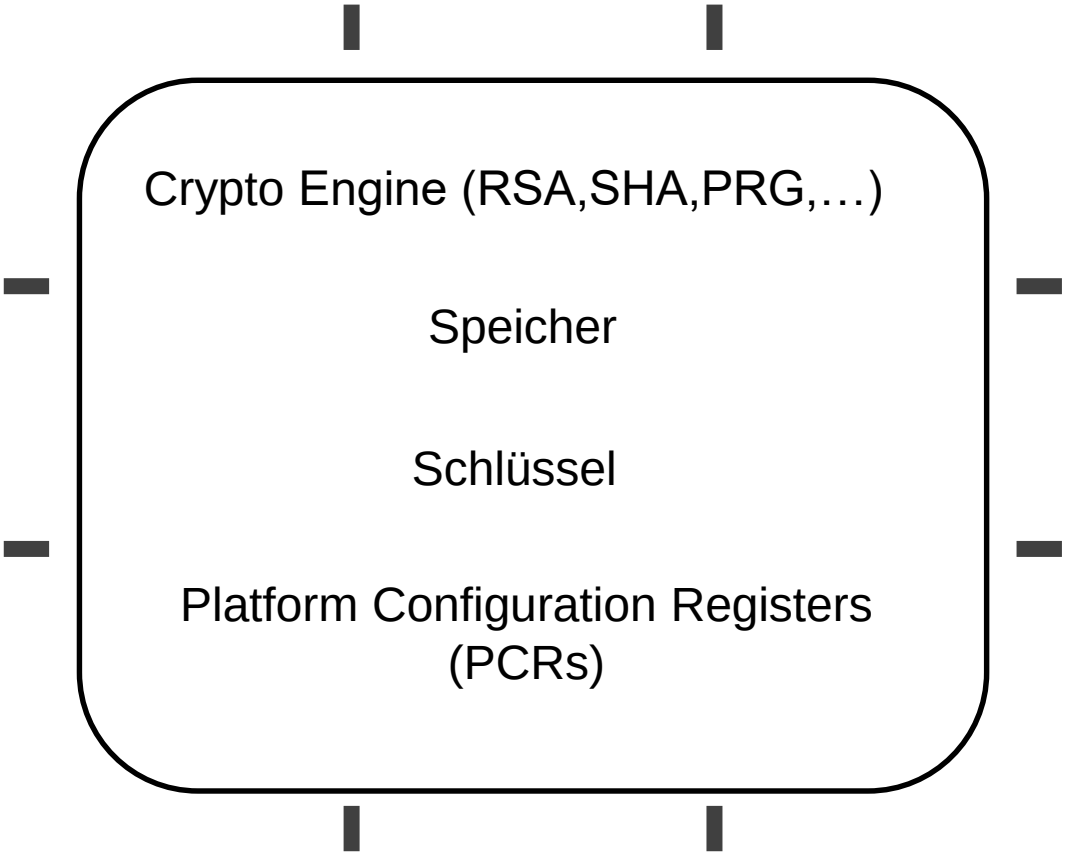
spezielle Crypto auf Intel-Chips ab Skylake,
um Hardware-geschützte Container („Enklaven“) zu erzeugen

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Trusted Platform Module (TPM)



vertrauenswürdige Komponenten in spezieller *zusätzlicher* HW



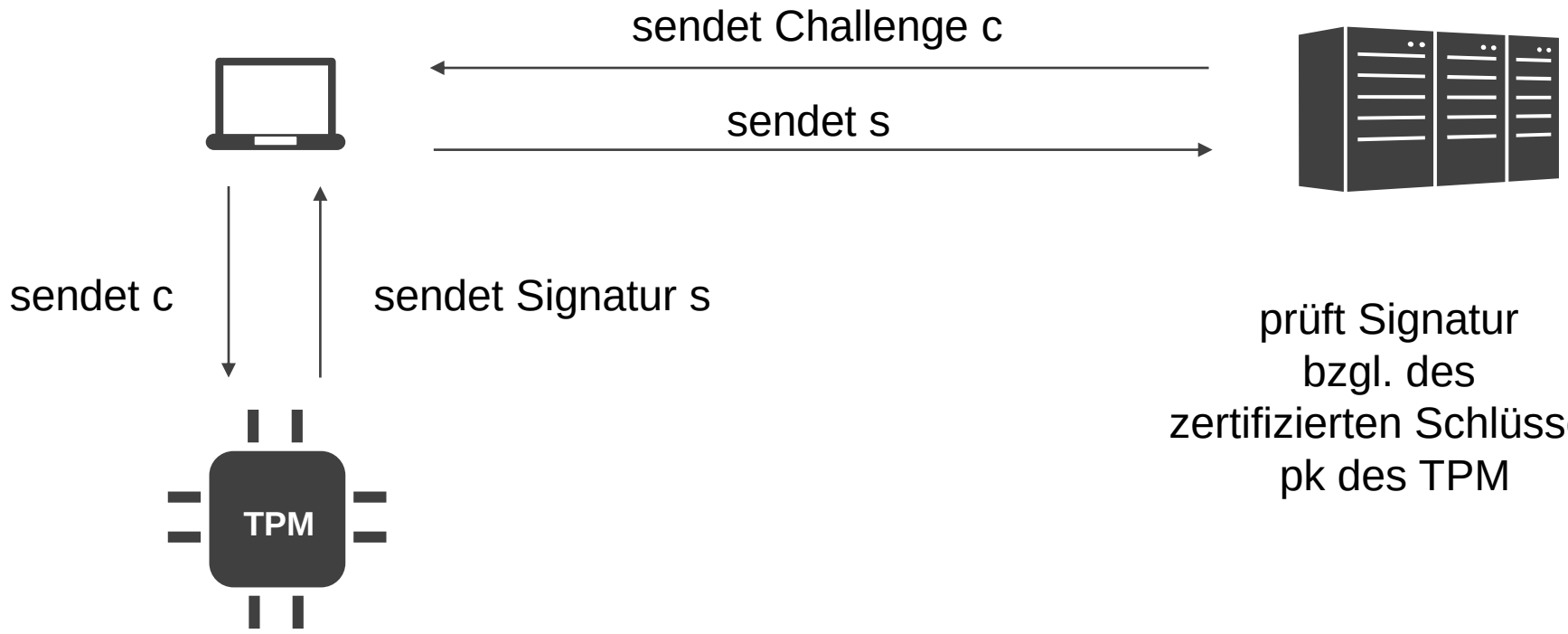
PCRs enthalten (Hashwerte der) SW/HW-Konfiguration des Systems

Messungen für PCRs werden von allen Komponenten HW, OS, App,… vorgenommen

erlauben es, Zustand durch Dritte prüfen zu lassen (Attestation)

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Allgemeines Prinzip der Attestation



→ „Embedded Systems Security“

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Testen und Verifizieren

→ „Einführung in Software Engineering“

→ „Formale Methoden in Softwareentwicklung“

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Testen vs. Verifizieren

Testet, ob SW/HW für bestimmte Eingaben korrekt

unvollständig

Nachweis, dass SW/HW Spezifikation erfüllt

Korrektheitsbeweis gemäß Spezifikation

komplex

"Testing can only show the presence of errors, not their absence."

E.Dijkstra

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Arten des Testens

statisch (anhand des Programms, ohne Ausführung)	vs.	dynamisch (zur Laufzeit)
white-box (mit Quellcode)	vs.	black-box (ohne Quellcode)

Testen von Sicherheitseigenschaften:
kein funktionales Verhalten, sondern (abstrakte) C.I.A.-Eigenschaften

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

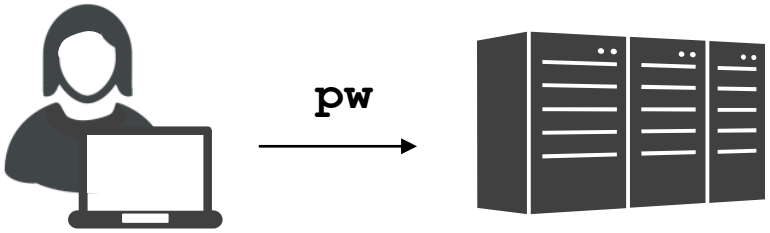
Security Testing

Vulnerability Testing	• Prüfe (automatisch) auf bekannte Schwächen • Beispiel: strcpy in Software
Penetration Testing	• simuliert (manuell und automatisch) Angriff • auch Ethical Hacking genannt • Beispiel: Führe Buffer Overflow-Angriff aus
Security Auditing	• Begutachtung gemäß Standards • Beispiel: ISO 27034-1
Risk Assessment	• Kosten-Nutzen-Analyse

idealerweise bereits im Software Development Life Cycle (SDLC) integriert

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity Cryptography & Complexity Theory Technische Universität Darmstadt www.cryptoplexity.de

Beispiel: Sichere Software >> Funktionale Software (I)



Angriff:
Versuche pw=‘a‘
Versuche pw=‘b‘
Versuche pw=‘c‘
usw. bis Antwort false==2

→ erster Buchstabe gefunden

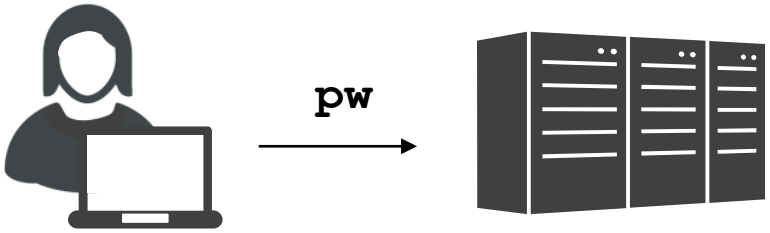
Fahre mit nächstem Buchstaben fort usw.

```
//Password checking in Pseudocode

pw   = get_password_from_user();
real = get_password_from_database():

false=0;
for i=1 TO real.length() do
{
   if pw[i] <> real[i] then
   {
      false=i;
      break;   //leave for-loop
   }
}
if (false == 0) then
{
   //allow access
}
else
{
   return_error_to_user(false);
}
```

Programm ist funktional korrekt

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Beispiel: Sichere Software >> Funktionale Software (II)



Seitenkanal-Angriff (vereinfacht!):

Versuche pw=‘a‘
Versuche pw=‘b‘
Versuche pw=‘c‘
usw. *bis Antwortzeit größer*

→ erster Buchstabe gefunden

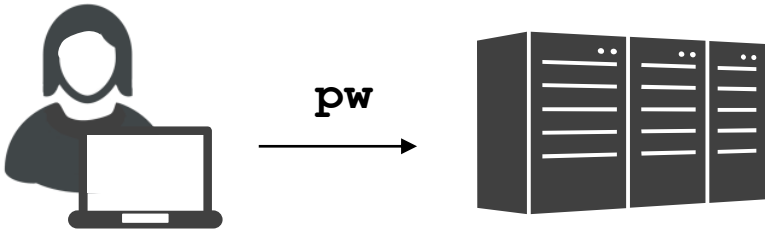
Fahre mit nächstem Buchstaben fort usw.

```
//Password checking in Pseudocode

pw   = get_password_from_user();
real = get_password_from_database():

false=0;
for i=1 TO real.length() do
{
   if pw[i] <> real[i] then
   {
      false=i;
      break;   //leave for-loop
   }
}
if (false == 0) then
{
   //allow access
}
else
{
   return_error_to_user();
}
```

Programm ist funktional korrekt

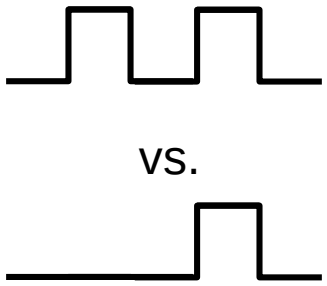
TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Beispiel: Sichere Software >> Funktionale Software (III)



Seitenkanal-Angriff (vereinfacht!):

Versuche pw='a'
Versuche pw='b'
Versuche pw='c'
usw.



bis Stromverbrauch für `false=i;` später

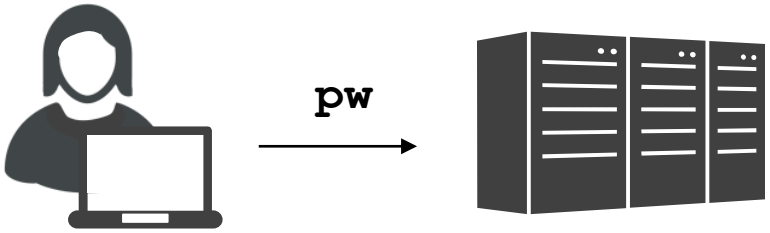
(z.B. bei Prüfung auf Smartcard)

```
//Password checking in Pseudocode

pw   = get_password_from_user();
real = get_password_from_database():

false=0;
for i=1 TO real.length() do
{
   if pw[i] <> real[i] then
   {
      false=i;
      //no break
   }
}
if (false == 0) then
{
   //allow access
}
else
{
   return_error_to_user();
}
```

Programm ist funktional korrekt

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Beispiel: Sichere Software >> Funktionale Software (IV)



Evtl. aber immer noch Seitenkanäle

z.B. wenn spekulative Ausführung

→ Spectre & Meltdown-Angriffe

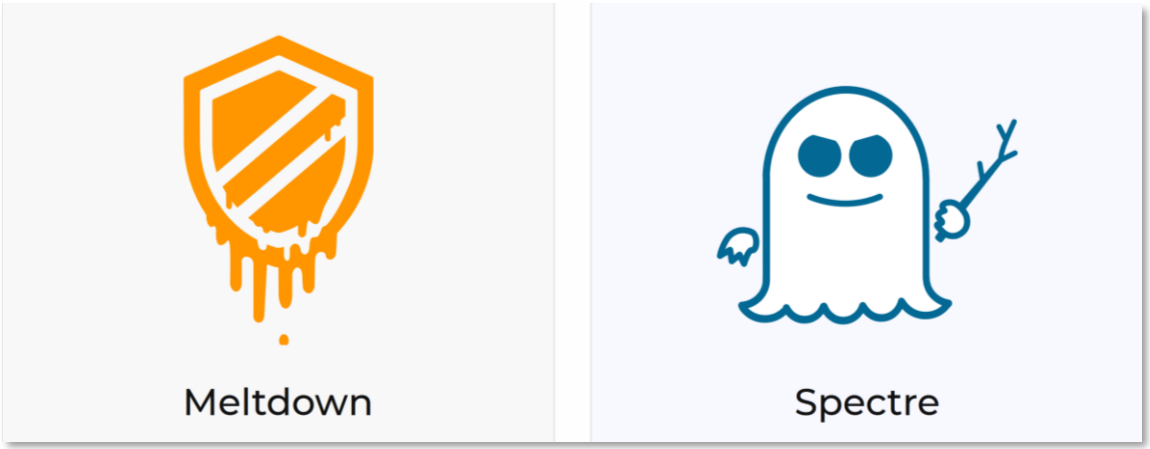
```
//Password checking in Pseudocode

pw   = get_password_from_user();
real = get_password_from_database():

false=0; fake=0;
for i=1 TO real.length() do
{
   if pw[i] <> real[i] then
      false=i;
   else
      fake=i;

}
if (false == 0) then
{
   //allow access
}
else
{
   return_error_to_user();
}
```

Programm ist funktional korrekt

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity

Spectre & Meltdown (Google & TU Graz, Januar 2018)

Angriffe auf spekulative Ausführungen von Code



Quelle:
www.meltdownattack.com

überbrückt Grenze zwischen Programm und Betriebssystem	überbrückt Grenze zwischen verschiedenen Programmen
(vor allem Kernel Memory auf Intel-Prozessoren)	(Cache-Informationen auf vielen Prozessoren)
einfacher durchzuführen	schwieriger zu verhindern

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

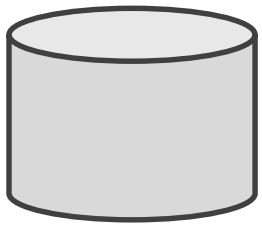
Quelle: Kocher et al: Spectre Attacks: Exploiting Speculative Execution, 2018

Spectre (vereinfacht!)

IP ⟶

```
if (x < array1_size)
   y = array2[array1[x] * 4096];
```

Prozessor holt evtl. Wert schon in Cache, bevor **`if`** ausgewertet (spekulative Ausführung für Geschwindigkeitsgewinn)



Cache

1. **Angreifer trainiert Branch-Predictor mit Ausführungen mit „guten“ Werten für `x`**
2. **Angreifer führt Programm mit “schlechtem“ `x` aus, Prozessor lädt Daten außerhalb des Bereichs spekulativ in Cache**
3. **Angreifer führt Cache-Angriff (z.B. Antwortzeiten) aus, um Daten zu erhalten**

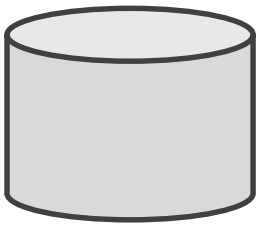
TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Quelle: Lipp et al, Meltdown: Reading Kernel Memory from User Space, 2018

Meltdown (vereinfacht!)

```
raise_exception();
// the line below is never reached
access(probe_array[data * 4096]);
```

Prozessor holt evtl. Wert schon in Cache (out-of-order execution);
Autorisierungstest für Kernel Memory wird noch nicht ausgeführt



Cache

Angreifer führt Cache-Angriff (z.B. Antwortzeiten) aus, um Daten zu erhalten

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Effekte

```
f94b76f0: 12 XX e0 81 19 XX e0 81 44  6f 6c 70 68 69 6e 31 |........Dolphin1|
f94b7700: 38 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |8...............|
f94b7710: 70 52 b8 6b 96 7f XX XX XX  XX XX XX XX XX XX XX |pR.k............|
f94b7720: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7730: XX XX XX XX 4a XX XX XX XX  XX XX XX XX XX XX XX |....J...........|
f94b7740: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7750: XX XX XX XX XX XX XX XX XX  XX e0 81 69 6e 73 74 |............inst|
f94b7760: 61 5f 30 32 30 33 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |a_0203..........|
f94b7770: 70 52 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |pR.}(...........|
f94b7780: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7790: XX XX XX XX 54 XX XX XX XX  XX XX XX XX XX XX XX |....T...........|
f94b77a0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b77b0: XX XX XX XX XX XX XX XX XX  XX XX XX 73 65 63 72 |............secr|
f94b77c0: 65 74 70 77 64 30 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |etpwd0..........|
f94b77d0: 30 b4 18 7d 28 7f XX XX XX  XX XX XX XX XX XX XX |0..}(...........|
f94b77e0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b77f0: XX XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
f94b7800: e5 e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 |................|
f94b7810: 68 74 74 70 73 3a 2f 2f 61  64 64 6f 6e 73 2e 63 |https://addons.c|
f94b7820: 64 6e 2e 6d 6f 7a 69 6c 6c  61 2e 6e 65 74 2f 75 |dn.mozilla.net/u|
f94b7830: 73 65 72 2d 6d 65 64 69 61  2f 61 64 64 6f 6e 5f |ser-media/addon_|
```

Listing (4) Memory dump of Firefox 56 on Ubuntu 16.10 on a Intel Core i7-6700K disclosing saved passwords.

Quelle: Lipp et al, Meltdown: Reading Kernel Memory from User Space, 2018

Firefox-Speicher
inklusive Passwort-Daten

Gegenmaßnahmen:
Kernel-Address-Isolation;
keine spekulative Ausführung

Performance-Verlust von ca. 5%-20%

ständig neue Varianten der Angriffe

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Was ist Sandboxing?

Diskutieren Sie Nachteile von Ansätzen, bei denen man vertrauenswürdige Hardware-Komponenten hat.

Nennen Sie Unterschiede zwischen Intels SGX-Technologie und dem TPM-Ansatz.

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Was Sie gelernt haben sollten

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de

Unterschied Würmer, Viren, Trojaner

Funktionsweise von Virenscannern

Buffer Overflows

Gegenmaßnahmen zu Buffer Overflows

Return-Oriented Programming

Isolation und Sandboxing

Trusted Platform Modules

Spectre & Meltdown

TECHNISCHE UNIVERSITÄT DARMSTADT
Cryptoplexity
Cryptography & Complexity Theory
Technische Universität Darmstadt
www.cryptoplexity.de