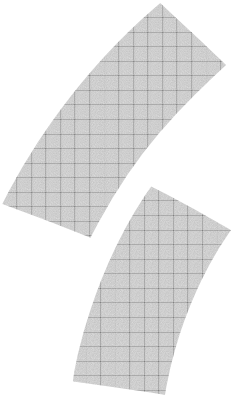
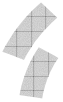


2003

Software-Schwachstellen

Tobias Klein (tk@cirosec.de)

cirosec



Software-Schwachstellen - Überblick

- Verschiedene Klassen: Buffer Overflows, Format-String-Schwachstellen, Integer Overflows, XSS, ...
- Schwachstellen in Netzwerkdiensten oder -applikationen
  - →DoS, Remote-Kompromittierung
  - →Firewall (Paketfilter) bietet keinen Schutz
  - →Andere Maßnahmen erforderlich
- Äußerst umfangreiche Thematik

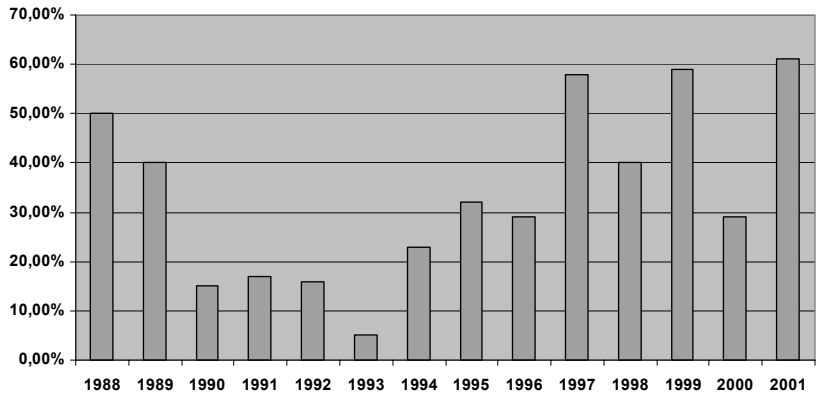
cirosec

Vortragsüberblick

- **Thema:** Buffer Overflows
  - Skizzierung der Ursachen/Funktionsweise
  - Gegenmaßnahmen
- **Ziel:** Überblick zur Buffer-Overflow-Problematik und eventuellen Gegenmaßnahmen
- **Grund:** fatale Auswirkungen, häufig (Schwachstelle Nr. 1)

cirosec

CERT-Advisories



cirosec

Ursachen

cirosec

Buffer Overflows - Ursache

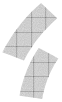
- Ursache: Programmierfehler

  →Unvermögen/Faulheit des Programmierers

- Speicherung der benutz.def. Inhalte in Puffern mit statischer Elementanzahl

- Fehlende oder fehlerhafte Längen-überprüfung der benutzerdefinierten Eingaben

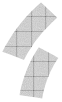
cirosec



Buffer Overflows - Ursache

- Betrifft: Die Programmiersprache C/C++
  - Performance ist oberstes Ziel
  - Bei Array- und Zeigerreferenzierungen keinerlei automatische Überprüfungen hinsichtlich der Einhaltung von Puffergrenzen
- Dies bleibt vielmehr dem Programmierer selbst überlassen.
- Darüber hinaus: Eine Reihe von „unsicheren" Bibliotheksfunktionen der libc

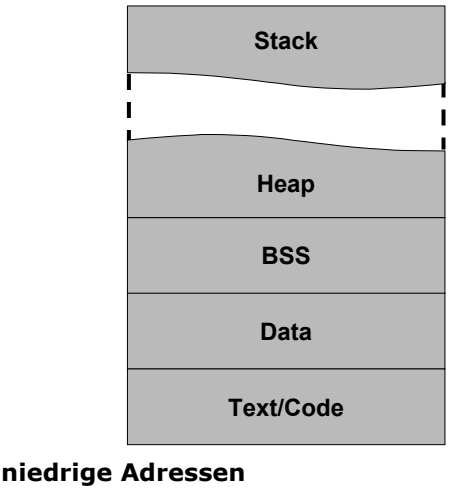
cirosec



Prozessspeicherlayout

cirosec

Prozessspeicherlayout



cirosec

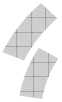
Verschiedene Bof-Generationen

- Verschiedene Generationen von Buffer Overflows
- Betreffen unterschiedliche Bereiche des Prozessspeichers

1. Generation: Klassische stack-basierte Buffer Overflows
2. Generation: Off-by-One Overflows und Frame Pointer Overwrites
3. Generation: BSS Overflows
4. Generation: Heap Overflows

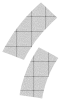
Evading Techniken: Return-into-Lib(c), Return-into-PLT, ...

cirosec



Klassische stack-basierte Buffer Overflows

cirosec

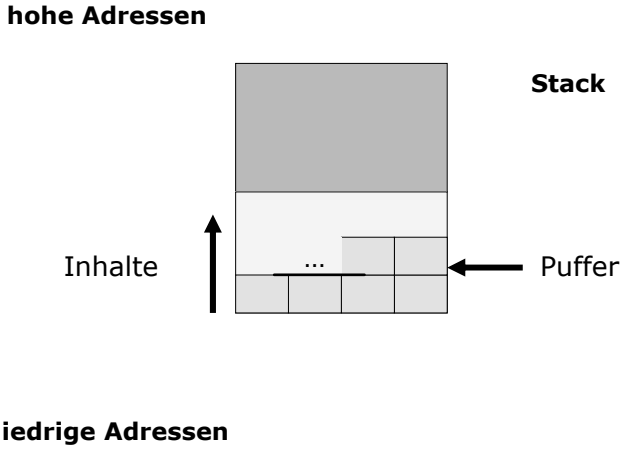


Klassische stack-basierte Bofs

- IA-32
- Stack wächst in Richtung niedrigerer Adresswerte
- Pufferinhalte wachsen in Richtung höherer Adresswerte

cirosec

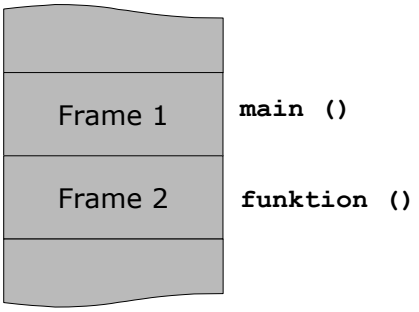
Klassische stack-basierte Bofs



cirosec

Klassische stack-basierte Bofs

- Jede einzelne Funktion/Prozedur eines Programms besitzt einen eigenen Bereich auf dem Stack → *Stack Frame*.



cirosec

Klassische stack-basierte Bofs

- Stack Frames werden durch stack-interne Informationen verwaltet.
- Rücksprungadresse (RET): Dient dazu, um von einer Unterfunktion an die richtigen Stelle in die aufrufende Funktion zurückzukehren.
- Saved Frame Pointer (SFP): Dient zur Sicherung der momentanen Stack-Umgebung vor einem Sprung in eine Unterfunktion.

cirosec

Buffer Overflows – Prinzip

```
void
funktion (char *args)
{
        char    buff[512];
        strcpy (buff, args);
}

int
main (int argc, char *argv[])
{
        if (argc > 1)
        {
                funktion (argv[1]);
        } else
                printf ("Kein Argument!\n");

        return 0;
}
```

Puffer mit statischer Elementanzahl

cirosec

Buffer Overflows – Prinzip

```
void
funktion (char *args)
{
        char    buff[512];
        strcpy (buff, args);
}

int
main (int argc, char *argv[])
{
        if (argc > 1)
        {
                funktion (argv[1]);
        } else
                printf ("Kein Argument!\n");

        return 0;
}
```

benutzerdefinierte Eingabe

cirosec

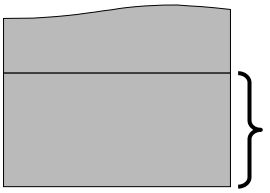
Buffer Overflows – Prinzip

```
void
funktion (char *args)
{
        char    buff[512];
        strcpy (buff, args);
}

int
main (int argc, char *argv[])
{
        if (argc > 1)
        {
                funktion (argv[1]);
        } else
                printf ("Kein Argument!\n");

        return 0;
}
```

cirosec

Buffer Overflows – Prinzip

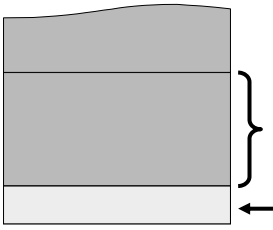
```
int
main (int argc, char *argv[])
```



cirosec

Buffer Overflows – Prinzip

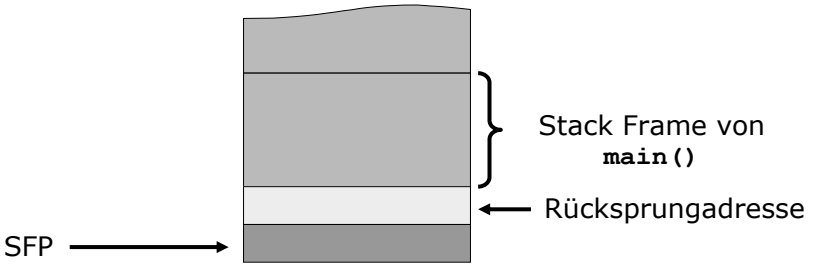
```
funktion (argv[1]);
```



cirosec

Buffer Overflows – Prinzip

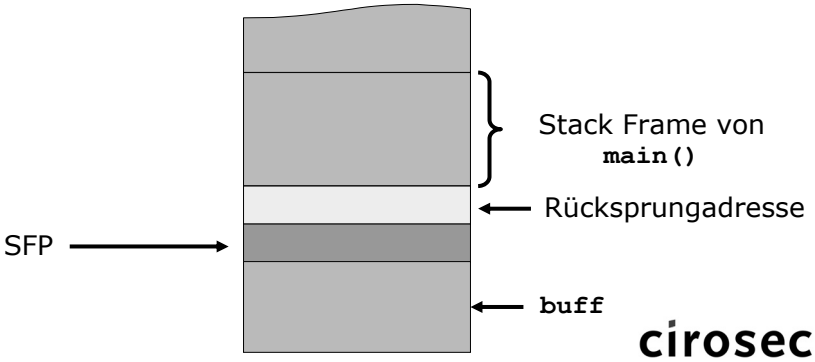
```
void
funktion (char *args)
```



cirosec

Buffer Overflows – Prinzip

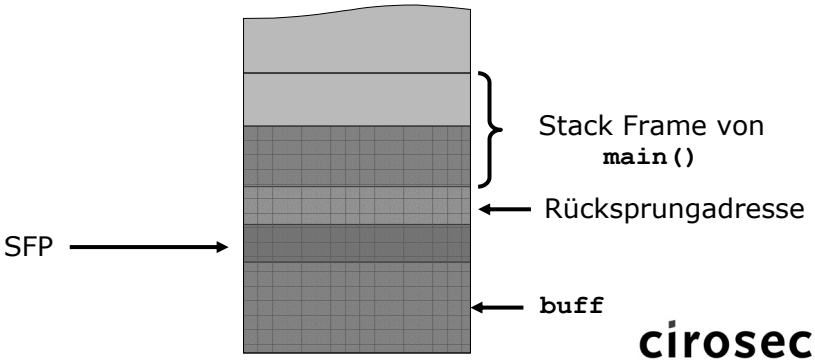
```
char    buff[512];
```



cirosec

Buffer Overflows – Prinzip

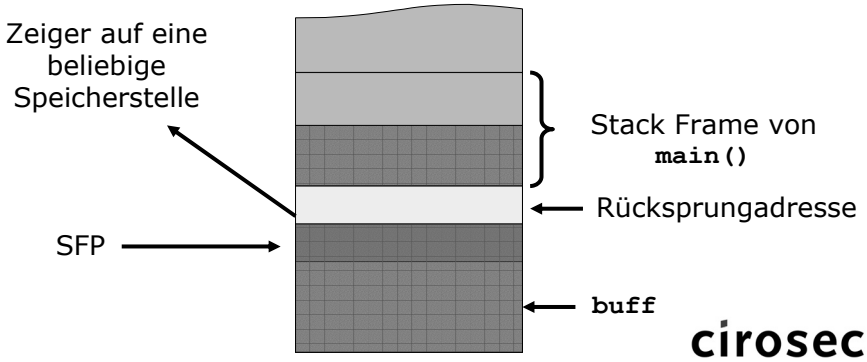
```
strcpy (buff, args);
```



cirosec

Buffer Overflows – Prinzip

```
strcpy (buff, args);
```



cirosec

Buffer Overflows – Prinzip

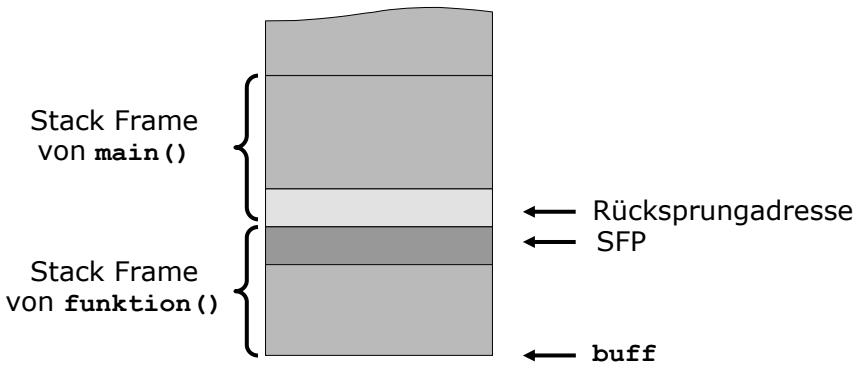
- Gezielte Manipulation der Rücksprungadresse

→ Möglichkeit zur beliebigen Manipulation des Programmflusses

- Ausführung bereits vorhandenen Programmcodes (Stichwort: Text-Bereich, Bibliotheken)
- Ausführung eingeschleusten Programmcodes

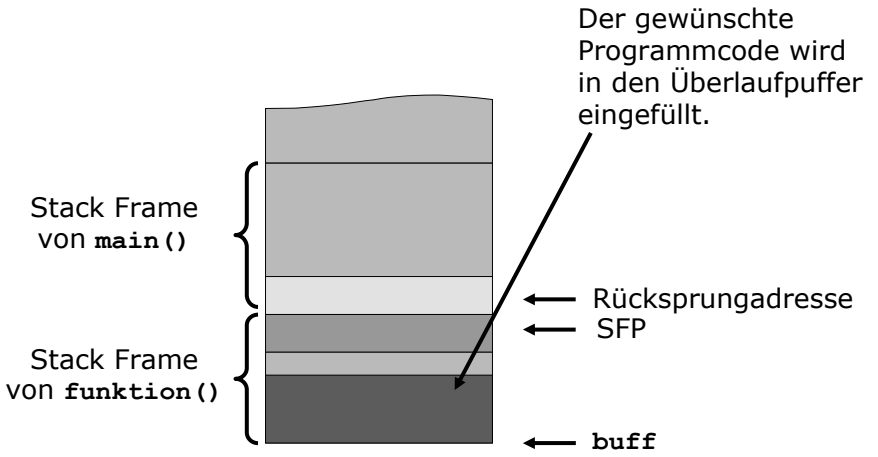
cirosec

Buffer Overflows – Prinzip



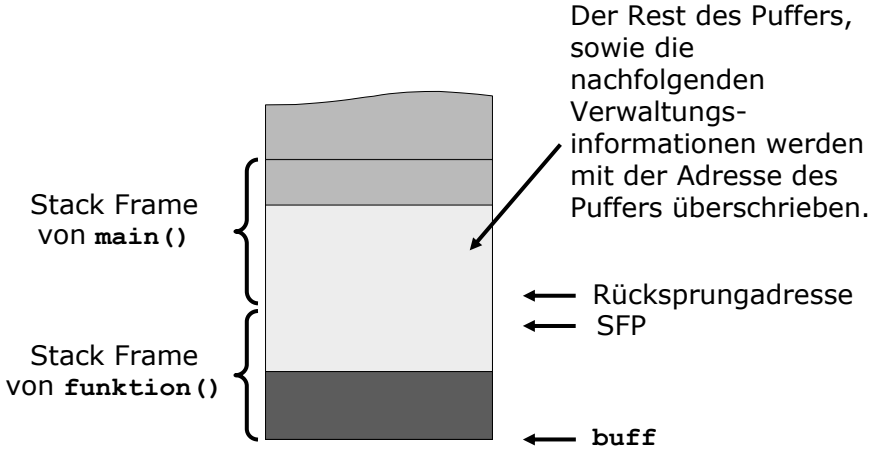
cirosec

Buffer Overflows – Prinzip



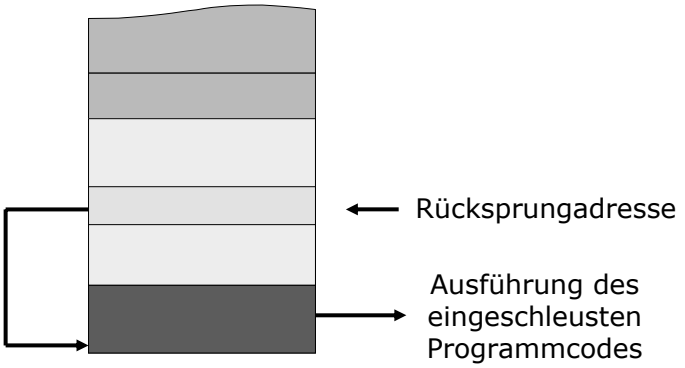
cirosec

Buffer Overflows – Prinzip



cirosec

Buffer Overflows – Prinzip



cirosec

Buffer Overflows – Prinzip

- Eingeschleuster Programmcode:
  - In der Regel ein Shellcode
  - Öffnen einer Shell mit den Rechten des kompromittierten Programmes
- ERGEBNIS: User- oder root-Shell

cirosec

Off-By-One

cirosec

Off-by-One

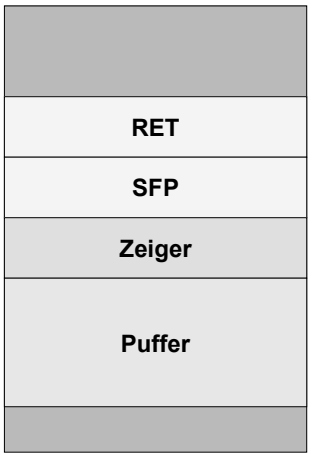
- Ein-Byte-Überlauf

```
[...]
        char   buff[256];
        int    i;
        for (i = 0; i <= 256; i++)
                buff[i] = eingabe[i];
[...]
```

- C/C++: Erstes Element eines Arrays → Index 0

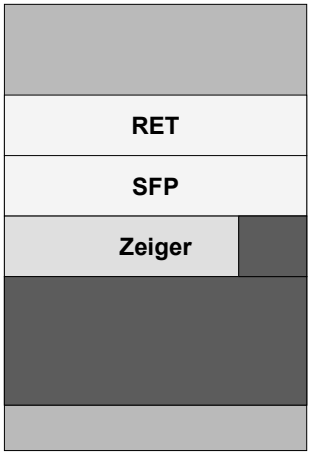
cirosec

Off-by-One



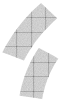
cirosec

Off-by-One



Kontrolle über das LSB des Zeigers

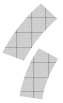
cirosec



Off-by-One

- Funktionszeiger
- Zeiger auf Dateien
- Jump-Konstrukte (`setjmp(3)/longjmp(3)`)
- Boolean-Werte
- ...

cirosec 



Frame Pointer Overwrite

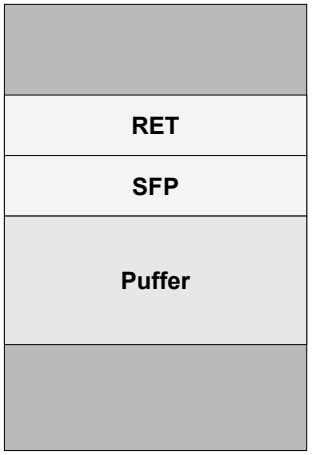
cirosec 

Frame Pointer Overwrite

- Spezielle Art von Off-By-One
- Der Überlaufpuffer befindet sich unmittelbar vor dem SFP
- Manipulation des LSB des SFP

cirosec

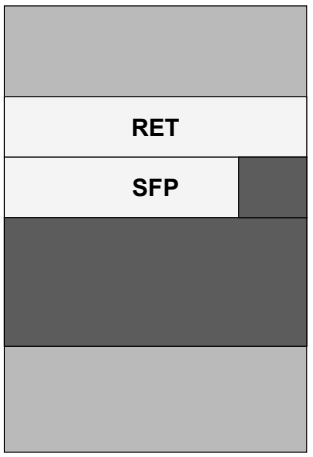
Off-by-One



Stack

cirosec

Off-by-One



Stack

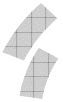
Kontrolle über das LSB des SFP

cirosec

Weitere Overflows

- BSS Overflows
  - Betrifft uninitialisierte globale/statische Variablen
  - Keine Verwaltungsinformationen
  - Aber → Zeiger ;)
  - Auch Off-By-Ones

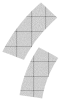
cirosec



Weitere Overflows

- Heap Overflows
  - Betrifft dynamisch allokierte Speicherblöcke (`malloc(3)`)
  - Verwaltungsinformationen werden zwischen den Daten auf dem Heap abgelegt ;)

cirosec



Gegenmaßnahmen

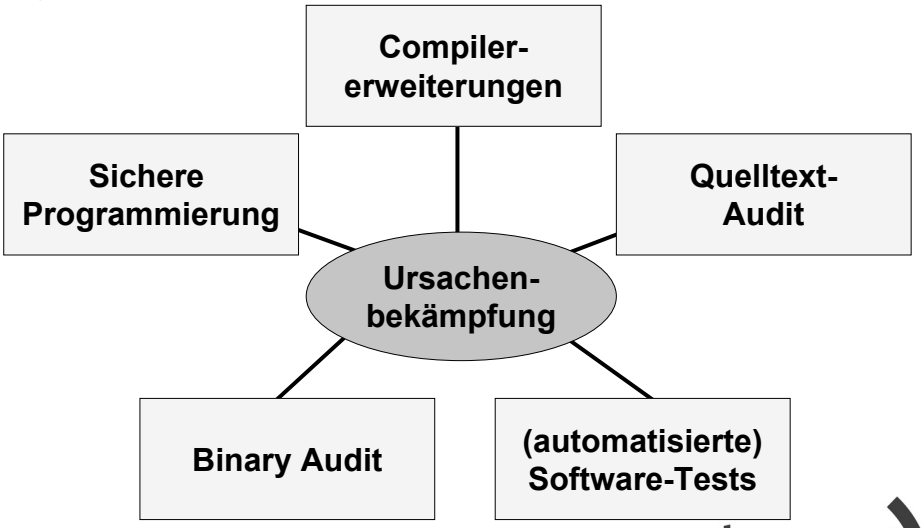
cirosec

Gegenmaßnahmen

- Drei Lösungsansätze
  - Ursachenbekämpfung (Programmierer)
  - Patchen (Admins)
  - Bekämpfung der Auswirkungen (Admins)

cirosec

Gegenmaßnahmen - Programmierer



cirosec

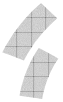
Sichere Programmierung

cirosec

Sichere Programmierung

- Vermeidung der Programmierfehler, welche zu Buffer-Overflow-Schwachstellen führen
- Durchsetzung entsprechender Programmier-richtlinien
- Sicherheitsbetonter Software-Entwicklungsprozess
- Alternative Programmiersprache (Java, Ada, etc.)?

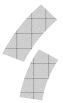
cirosec



Sichere Programmierung

- Vermeidung unsicherer Bibliotheks-funktionen
  - **gets(3), strcpy(3), strcat(3), [v]sprintf(3), getenv(3), read(3), [v,f,s,vs,vf]scanf(3), getchar(3), [f]getc(3), …**

cirosec



Sichere Programmierung

- Verwendung (vermeintlich) „sicherer" Alternativen
  - **fgets(3), strncpy(3), strncat(3), {v}snprintf(3), ...**
- Vorsicht: bergen ihre eigenen Fallstricke
  - Z.B. fehlende NULL-Terminierung bei **strncpy(3)**

cirosec

Sichere Programmierung

- **`strncpy(3)`** – Ermöglicht Längenprüfung, aber: fehlende Null-Terminierung

Falsch:

```
strncpy (dest, src, BUFFERGR);
```

Richtig:

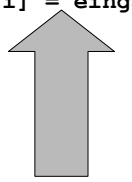
```
strncpy (dest, src, BUFFERGR - 1);
dest[BUFFERGR - 1] = '\0';
```

cirosec

Sichere Programmierung

- Fehlerhafte Schleifenkonstruktionen:

```
[...]
        char   buff[256];
        int    i;
        for (i = 0; i <= 256; i++)
                buff[i] = eingabe[i];
[...]
```



cirosec

Quelltext-Audit (manuelles Vorgehen)

cirosec

Quelltext-Audit

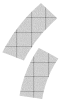
- Manuelle, zeilenweise Überprüfung des Quellcodes
  - Puffer statischer Elementgröße ausfindig machen

Beispiele:

```
char  buff[128];
static char  buff[128];
buff = malloc (128);
…
```

  - Weitere Verarbeitung dieser Puffer auf Schwachstellen überprüfen

cirosec



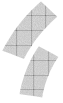
Alternative Programmiersprachen

- Java, Perl, PHP etc.
- Sind nicht anfällig für Bofs

```
java.lang.ArrayIndexOutOfBoundsException
```

- Besitzen jedoch ihre jeweils eigenen Probleme/Schwachstellen
- Weiteres Problem: Einbindung nativen Codes

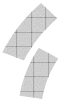
cirosec



Alternative Programmiersprachen

- Skriptsprachen in der Web-Entwicklung sind häufig nur Container für native Libs
- Java
  - JNI Schnittstelle
  - Bof in nativem Code → das wars
  - JVM ist selbst in C geschrieben

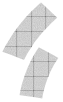
cirosec



Java – Bof in nativem Code

```
An unexpected exception has been
detected in native code outside the VM.

Unexpected Signal : 11 occurred at
PC=0x41414141
```

cirosec



(automatisierte) Software-Tests

cirosec

(automatisierte) Software-Tests

Zwei Analyseverfahren:

- *Statische Analyse* → Überprüfung des Quellcodes (Source Code Analyzer)
  - *Lexikalische Analyse* (hinsichtlich der Bedeutungseinheit, ohne Rücksicht auf den Zusammenhang)
  - *Semantische Analyse* (detailliertere Rückschlüsse mittels Datenflussanalyse)
- *Dynamische Analyse* → Überwachung der Programmausführung (Tracer, Debugger)

cirosec

(automatisierte) Software-Tests

Statische Analyse:

- Source Code Analyzer - *Lexikalische Analyse*
- Werkzeuge:
  - **`grep(1)`** → viele False Positives
  - *flawfinder* (www.dwheeler.com/flawfinder/)
  - *Rats* (www.securesoftware.com/rats.php)
  - *ITS4* (www.cigital.com/its4/)
  - …

cirosec

(automatisierte) Software-Tests

- Beispiel: *flawfinder*
- Fehlerhaftes Programm:

```
01  #include <string.h>
02
03  int
04  main (int argc, char *argv[])
05  {
06     char   buff[24];
07
08     if (argc > 1)
09           strcpy (buff, argv[1]);
10     return 0;
11  }
```

cirosec

(automatisierte) Software-Tests

- Ausgabe von *flawfinder:*

```
[user]$ flawfinder bsp.c
Flawfinder version 0.21, (C) 2001 David A. Wheeler.

[…]

strcpy.c:9 [4] (buffer) strcpy: does not check for buffer
overflows. Consider using strncpy or strlcpy.

[…]
```

cirosec

(automatisierte) Software-Tests

Statische Analyse:

- Source Code Analyzer – *Semantische Analyse*
- Werkzeuge:
  - *Splint* (www.splint.org)
  - Compiler-Optionen (**`-Wall`**)
  - …

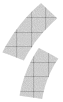
cirosec

Splint

- Splint überprüft, „[...] that the source code is consistent with the properties implied by annotations." (Splint Manual).
- Bestimmte Annahmen bzw. Absichten werden durch *Annotations* spezifiziert und anschließend überprüft

```
void /*@alt char * @*/
  strcpy (/*@unique@*/ /*@out@*/ /*@returned@*/
char *s1, char *s2)
     /*@modifies *s1@*/
     /*@requires maxSet(s1) >= maxRead(s2) @*/
     /*@ensures maxRead(s1) == maxRead (s2) /\
maxRead(result) == maxRead(s2) /\ maxSet(result) ==
maxSet(s1); @*/;
```

cirosec

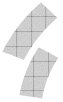


(automatisierte) Software-Tests

Dynamische Analyse:

- Tracer
- Entsprechende Werkzeuge:
  - Purify (www.rational.com)
  - ElectricFence (ftp://ftp.perens.com/pub/ElectricFence/)
  - Valgrind (http://developer.kde.org/~sewardj/)
  - **`strace(3)`, `ltrace(3)`**
  - …

cirosec



Innovative Ideen ;)

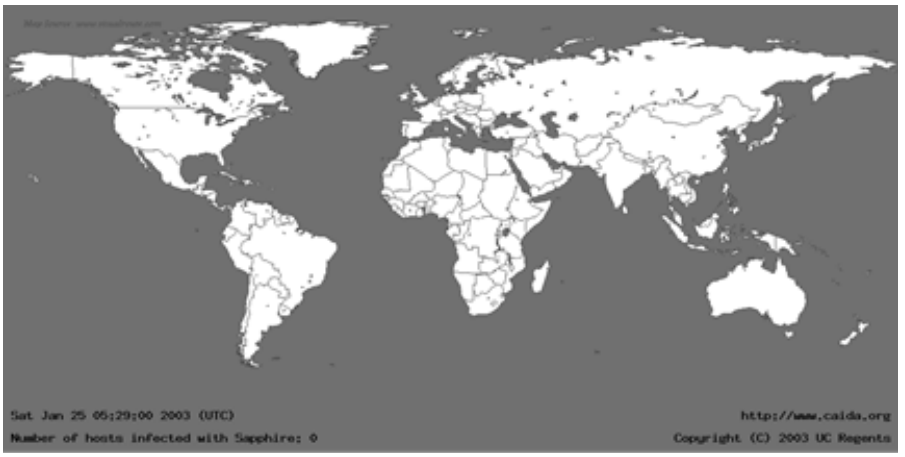
„At Microsoft we halted development on several key products and invested more than $100 million to evaluate our existing software for security issues, and to train our developers to build security into our future products from the ground up."

- Bill Gates, 23. Jan. 2003

http://www.microsoft.com/presspass/ofnote/01-03davos.asp

cirosec

… 2 Tage später …



cirosec

Sapphire-Wurm – Buffer Overflow in MSSQL

cirosec

Binary Audit

cirosec

Binary Audit

Zwei Ansätze:

- Fault Injection
- Überprüfung des Programm-Disassembly's (Reverse Engineering)

cirosec

Binary Audit – Fault Injection

- Gezielte Übergabe von unerwarteten Eingaben an die Applikation
- Beobachtung der entsprechenden Reaktion
- Klassisch: Segmentation Fault, Coredump

```
$ ./binary `perl -e 'print "A"x10000'`
Segmentation fault
```

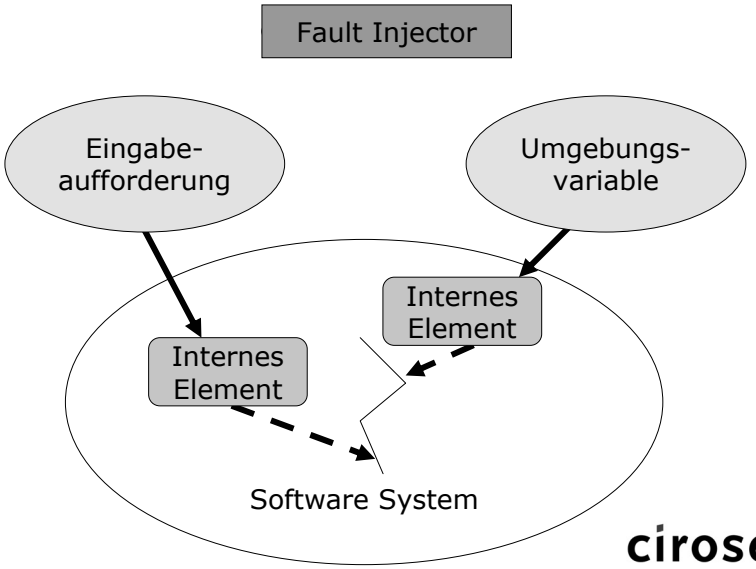
cirosec

Binary Audit – Fault Injection

- Netzwerkbasierte Werkzeuge:
  - Hailstorm (www.cenzic.com)
  - Spike (http://www.immunitysec.com/spike.html)
- Hostbasierte Werkzeuge:
  - BFBTester (bfbtester.sourceforge.net)
  - Sharefuzz (sourceforge.net/projects/sharefuzz/)

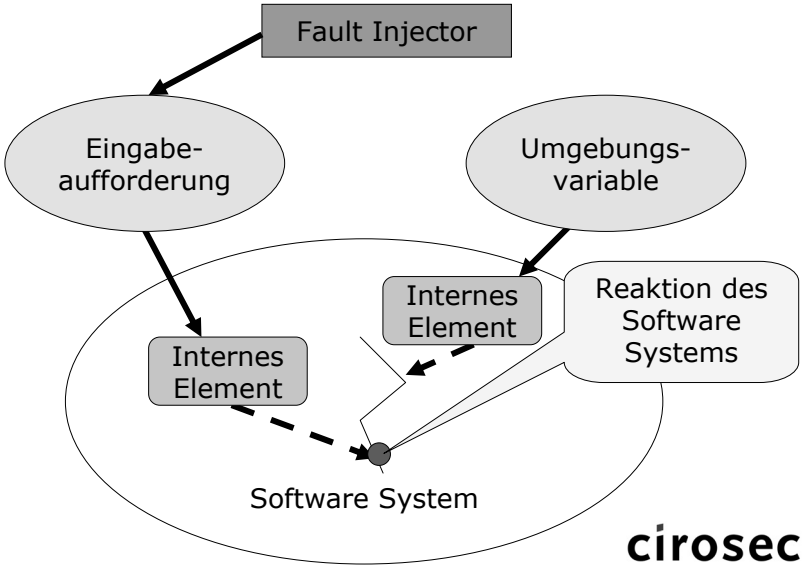
cirosec

Fault Injection



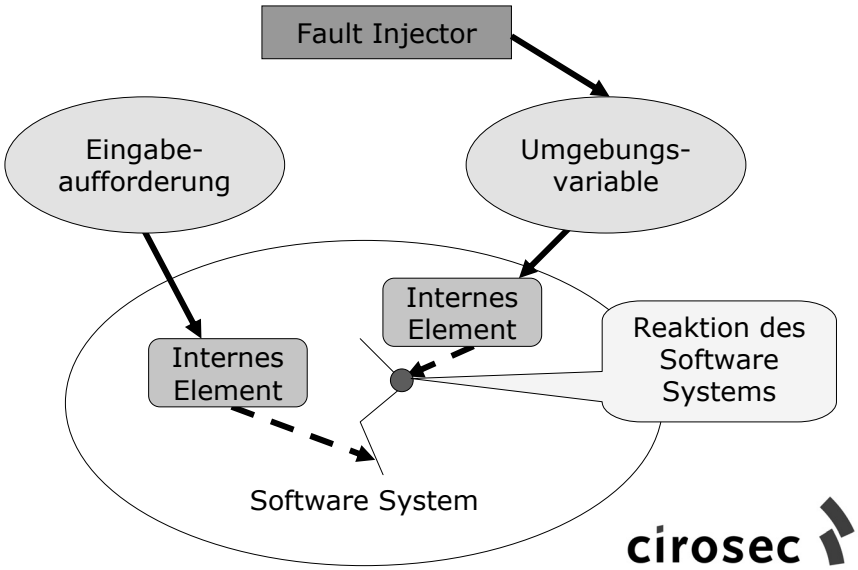
cirosec

Fault Injection



cirosec

Fault Injection

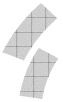


cirosec

Binary Audit – Reverse Engineering

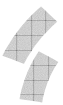
- Lediglich Zugriff auf das Binary
- Proprietäre Software
- Werkzeug/Disassembler der Wahl:
  - IDA Pro (www.datarescue.com/idabase/)

cirosec



Binary Audit – Reverse Engineering

```
.text:00401000                  push    ebp
.text:00401001                  mov     ebp, esp
.text:00401003                  sub     esp, 18h
.text:00401006                  mov     eax, [ebp+arg_4] ; Source Buffer
.text:00401009                  push    dword ptr [eax+4] ; char *
.text:0040100C                  lea     eax, [ebp+var_18] ; Destination Buffer
.text:0040100F                  push    eax             ; char *
.text:00401010                  call    _strcpy
```



Compiler-Erweiterungen

Compiler-Erweiterungen

Diverse Implementationen:

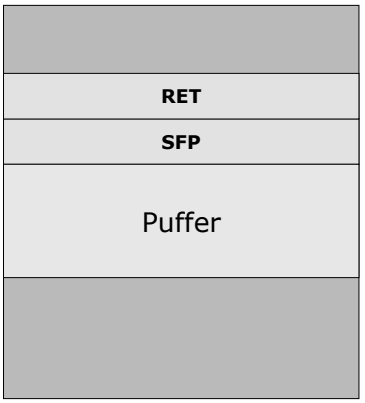
- Komplettes Bounds-Checking (http://www-ala.doc.ic.ac.uk/~phjk/BoundsChecking.html)
- StackGuard (www.immunix.org), /GS-Option (Visual C++ 7) → Canary
- StackShield (http://www.angelfire.com/sk/stackshield/)
- SSP (http://www.trl.ibm.com/projects/security/ssp/)

cirosec

Compiler-Erweiterungen

Exemplarische Funktionsweise (StackGuard):

Stack:

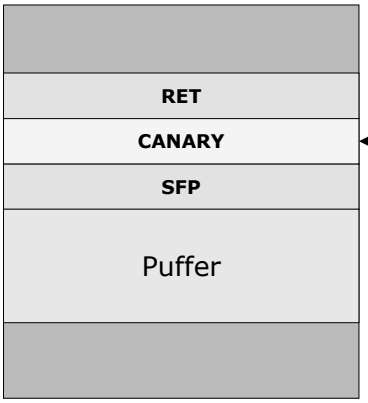


cirosec

Compiler-Erweiterungen

Exemplarische Funktionsweise (StackGuard):

Stack:



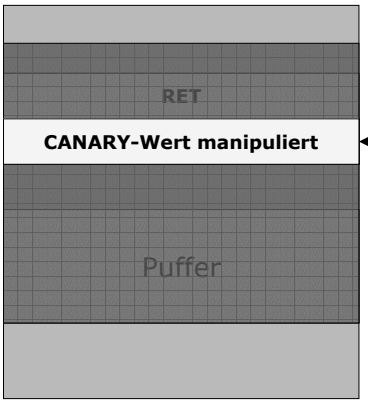
Ständige Überprüfung der Canary-Integrität

cirosec

Compiler-Erweiterungen

Exemplarische Funktionsweise (StackGuard):

Stack:

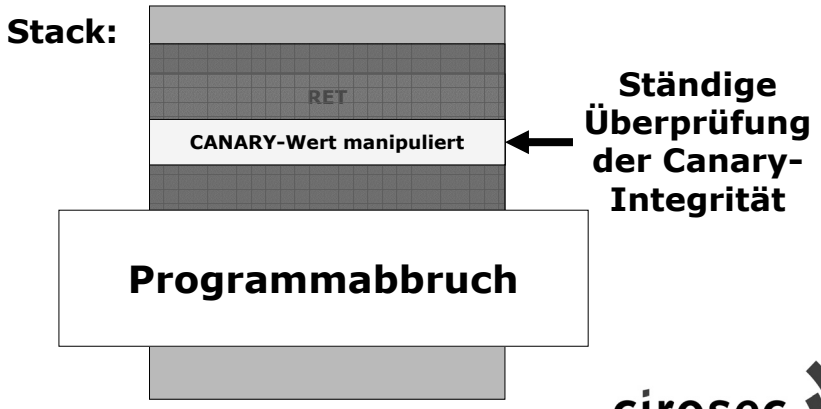


Ständige Überprüfung der Canary-Integrität

cirosec

Compiler-Erweiterungen

Exemplarische Funktionsweise (StackGuard):



cirosec

Gegenmaßnahmen – Admins

Patches

cirosec

Patches

Contra:

- Werden zu spät eingespielt
- Angreifer ist stets einen Schritt voraus
- Ablehnung gegen das Einspielen (n.t.a.r.s.)
- Neue Schwachstellen (Grund: Zeitdruck)

- Reaktive Zwangsmaßnahme

cirosec

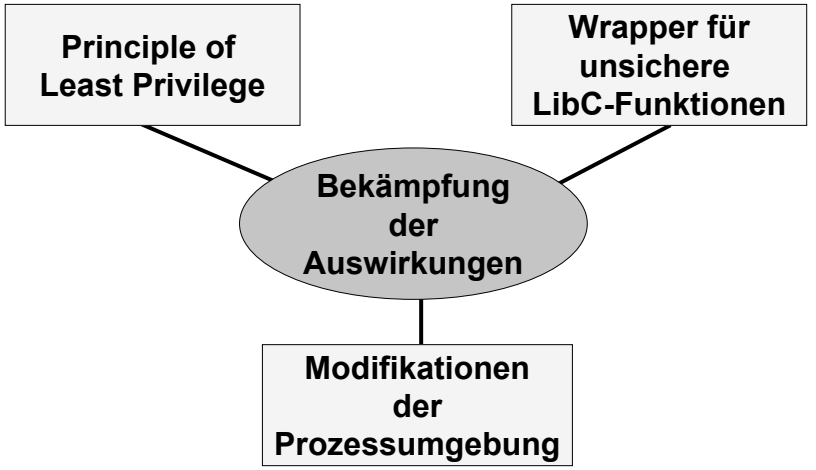
;-)

„Our recommendation now is the same as our recommendation a month ago, if you haven't patched your software, do so now."

– Microsoft's security program manager on the buffer overflow vulnerability in IIS that was exploited by the Code Red worm to acquire over 300,000 zombie machines to launch a distributed denial-of-service attack on the White House web site.

cirosec

Gegenmaßnahmen – Admins



cirosec

Wrapper für „unsichere“ Bibliotheksfunktionen

cirosec

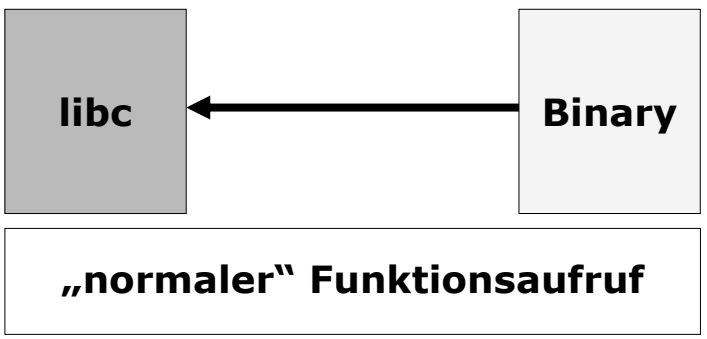
Wrapper für „unsichere" Bibliotheks-funktionen

- Beispielimplementation unter Linux:
  - *Libsafe* (www.research.avayalabs.com/project/libsafe/)
- Ersetzt potentiell „unsichere" Funktionen durch „sichere" Alternativen

cirosec

Wrapper für „unsichere" Bibliotheks-funktionen

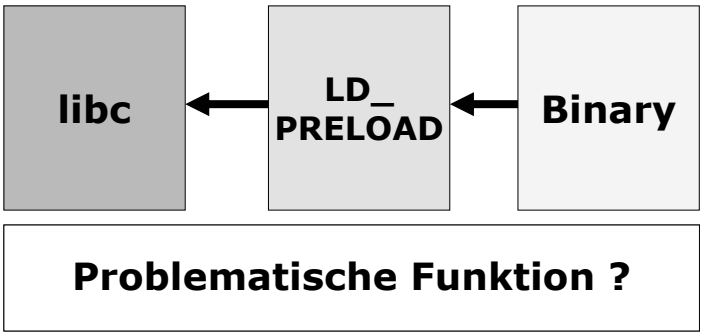
Funktionsweise:



cirosec

Wrapper für „unsichere“ Bibliotheks-funktionen

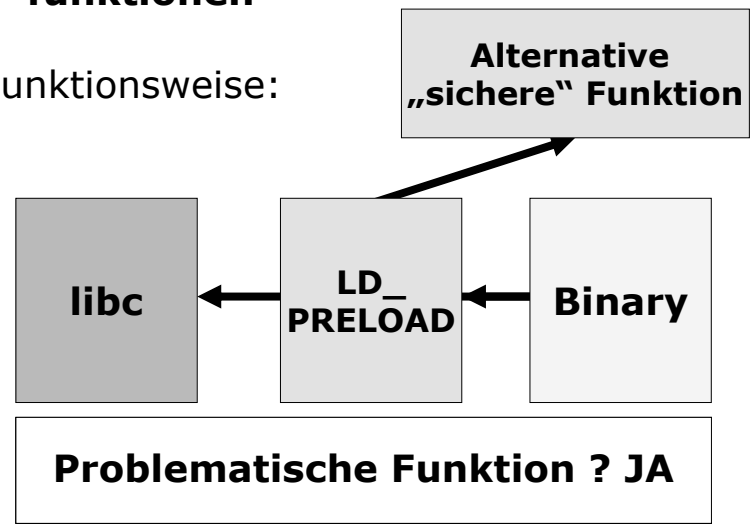
Funktionsweise:



cirosec

Wrapper für „unsichere“ Bibliotheks-funktionen

Funktionsweise:



cirosec

Modifikationen der Prozessumgebung

cirosec

Modifikationen der Prozessumgebung

- Techniken:
  - Nicht-ausführbare Speicherbereiche sowie Pages
  - Zufällige Bestimmung der Prozessspeicher-adresswerte
- Entsprechende Implementationen:
  - **Linux:** Openwall (www.openwall.com), PaX (http://pageexec.virtualave.net/)
  - **Solaris:** non-exec Stack
  - **Windows:** SecureStack (www.securewave.com)

cirosec

Principle of Least Privilege

cirosec

Principle of Least Privilege

Betriebssystemhärtung

- Proaktiv:
  - SUID/SGID (UNIX)
  - Angebotene Dienste
  - Dienst-Privilegien
- Reaktiv

  → Einspielen von Patches

cirosec

Principle of Least Privilege

Ebenfalls proaktiv:

- Secure Application Environments (SAE)
- Intrusion Prevention Systeme (IPS)
- Trusted Operating Systems (TOS)
- Sandbox-Systeme
- …

cirosec

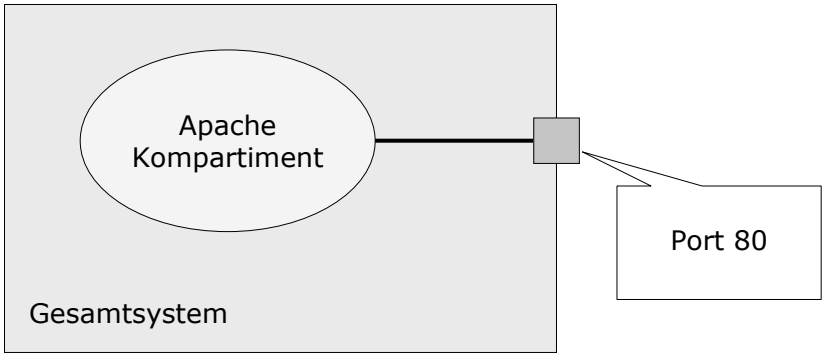
Principle of Least Privilege

- Aufsätze auf das Basis-Betriebssystem
- Einschränkung der Rechte einer/eines Applikation/Dienstes beim Zugriff auf OS-Ressourcen
  - Netzwerk
  - Dateien
  - andere Prozesse
- Ziele
  - Schadensbegrenzung innerhalb des Servers
  - Verhinderung weitergehender Angriffe

cirosec

Secure Application Environments (SAE)

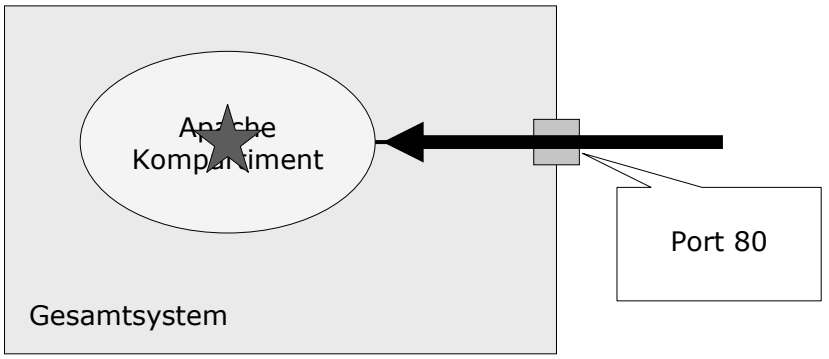
- Exemplarisch: *Pitbull Lx* (www.argus-systems.com)



cirosec

Secure Application Environments (SAE)

- Exemplarisch: *Pitbull Lx* (www.argus-systems.com)



cirosec

Weitere Lösungen/Produkte

- Stormwatch (www.okena.com)
- NSA SecureLinux (www.nsa.gov/selinux/)
- Trusted Solaris
- RSBAC (www.rsbac.org)
- HP Virtual Vault
- Lids (www.lids.org)
- Systrace (www.citi.umich.edu/u/provos/systrace/)
- …

cirosec

Fazit

- Firewall (Paketfilter) hilft nicht
- Völlige Beseitigung des Problems → fehlerfreie Software = Utopie ;)
- Kombination: Sicherheitsbetonte Software-Entwicklung + proaktive (Schutz-) Implementationen
- Sehr vielversprechende proaktive Ansätze: PaX, SAE, TOS, …

cirosec

cirosec

Ende

Fragen

cirosec