

Steffen Wendzel

---

Implementierung eines Userprofile  
Intrusion Detection-Systems in den  
OpenBSD-Kernel

## Inhaltsverzeichnis

<b>1 Anmerkung</b>	<b>3</b>
1.1 Version 0.2p1 . . . . .	3
<b>2 Zielpublikum</b>	<b>4</b>
<b>3 Einleitung</b>	<b>4</b>
<b>4 Intrusion Detection Systeme (IDS)</b>	<b>4</b>
<b>5 FUPIDS</b>	<b>5</b>
5.1 Features . . . . .	6
5.2 Attacker-Level . . . . .	6
5.2.1 Berechnung des Attacker-Levels . . . . .	7
5.3 Protokollierungsformat . . . . .	7
5.3.1 Neue Programme . . . . .	8
5.3.2 promiscuous Modus . . . . .	8
5.3.3 Attacker-Level . . . . .	8
5.3.4 listen()-Sockets . . . . .	9
5.4 Beispielüberwachung . . . . .	9
5.5 Installation . . . . .	9
5.6 Erweiterbarkeit . . . . .	12
5.7 Datei-Inodes . . . . .	12
5.8 Memory Allocation . . . . .	12
<b>6 Glossar</b>	<b>13</b>
6.1 Exploit . . . . .	13
6.2 Kernelspace . . . . .	13
6.3 Syscall . . . . .	13
6.4 Userspace . . . . .	13
<b>7 Literatur</b>	<b>13</b>

---

## 1 Anmerkung

Dieses Dokument ist die Dokumentation zum Abschlussprojekt meiner Ausbildung. Nähere Informationen zum Projekt gibt es unter [cdp.doomed-reality.org/fupids](http://cdp.doomed-reality.org/fupids).

### 1.1 Version 0.2p1

Diese Version, die einige Jahre später erschien<sup>1</sup>, beinhaltet zwei Tippfehlerkorrekturen.

(C) 2003,2004,2007 by Steffen 'cdp\_xe' Wendzel

cdp at doomed dash reality dot org / <http://cdp.doomed-reality.org>

---

<sup>1</sup>24. Februar 2007

## 2 Zielpublikum

Dieses Projekt richtet sich an erfahrene OpenBSD-Administratoren und -Security-Consultants, die auf Einzelsystemen eine Überwachung der Benutzertätigkeiten benötigen.

Dies setzt selbstverständlich auch ein hohes Maß an Grundwissen voraus, weshalb in dieser Dokumentation nicht alle Grundlagen im Detail besprochen werden.

## 3 Einleitung

Das Thema IT-Sicherheit ist in den letzten paar Jahren stetig mehr und mehr in den Mittelpunkt gerückt. Gründe dafür sind die ansteigende Anzahl von Angriffen auf Unternehmensnetzwerke und das steigende Bewusstsein der Administratoren für die Unsicherheit ihrer Systeme.

Heutzutage reicht es nicht mehr aus, einen Passwortschutz und einfache Zugriffsbeschränkungen über Rechte oder Access Control Lists zu vergeben.

- Für heutige Firmen-Netzwerke müssen strikte Security-Policies entwickelt werden.
- Netzwerkdienste sollten nur über Authentifizierungen, etwa via KerberosV nutzbar sein. Neben Kerberos, einer freien, vom MIT entwickelten, portablen (und damit auch in heterogenen Netzwerken einsetzbaren) Möglichkeit abgesehen gibt es für den Remote Access auch noch RADIUS oder TACACS+ auf CISCO Routern.
- Alle Systeme sollten den gleichen Patchlevel fahren, dieser wiederum sollte sich auf dem aktuellen Stand befinden.
- Datenverbindungen sollten verschlüsselt werden.
- usw.

## 4 Intrusion Detection Systeme (IDS)

Intrusion Detection Systeme haben die Aufgabe Systeme oder Netzwerke zu überwachen. Dabei unterscheidet man zwischen Host- und Netzwerk-basierten IDS, HIDS und NIDS.

Mit Hilfe verschiedener Techniken (Neuronale Netze, Fuzzy Logic, ...) können diese Systeme zwischen "normalen" und "abnormalen" Werten unterscheiden

(Anomalie-Erkennung). Einige IDS durchsuchen Werte auch nach vorgegebenen Regelsätzen, etwa das NIDS "Snort". Snort durchsucht den Netzwerktraffic nach den, vom Benutzer vorgegebenen Regelungen.

Der Vorteil eines Regelbasierten IDS ist, dass es keine Lernphase benötigt und weniger anfällig für Fehler ist. Der Nachteil besteht jedoch darin, dass keine Abnormalien von selbst erkannt werden können. Kennt der Security-Consultant einen Angriff nicht, kennt ihn auch das NIDS nicht. Die logische Folge ist, dass der Angriff nicht protokolliert werden kann.

## 5 FUPIDS

FUPIDS steht für "Fuzzy Userprofile Intrusion Detection System". Es basiert auf Fuzzy-Logic um im Kernel einem möglichst geringen Rechenaufwand (nur Addition und Subtraktion) durchführen zu müssen. Neuronale Netze benötigen Gleitkommaberechnungen die im Kernelspace vermieden werden müssen, Rechnungen mit float-Variablen (oder double) sind nicht möglich.

FUPIDS erstellt Benutzerprofile im Kernelspeicher. In diesen werden für jeden Benutzer die bereits gestarteten Programme protokolliert. Zudem wird ein "Attacker-Level" berechnet. Dieser Attacker-Level gibt an, wie hoch die Wahrscheinlichkeit dafür ist, dass ein Benutzer-Account übernommen wurde.

Der Gedanke dahinter ist folgender: Ein normaler Benutzer erzeugt ein Profil, verwendet hin und wieder neue Programme. Das Profil wird von FUPIDS verwaltet, hin und wieder neu eingetragene Programme bedeuten keine zu großen Anstiege im Attacker-Level, folglich bleibt der Benutzer sehr wahrscheinlich unbeachtet.

Wird ein Account jedoch übernommen, wird der Angreifer mit einer relativ hohen Wahrscheinlichkeit andere Programme verwenden, als der eigentliche Benutzer. Diese Programme bedeuten neue Einträge im Benutzerprofil und einen gewissen Anstieg im Attacker-Level, doch dazu später mehr.

Im Gegensatz zu anderen Betriebssystemen, gibt es unter Unix-Systemen eine breite Palette an Tools. Jeder Benutzer hat die Auswahl zwischen mehreren im POSIX-Standart festgelegten Tools. Hier ein Beispiel:

Der normale Benutzer verwendet unter der grafischen Oberfläche den Editor "kate". Der Angreifer könnte diesen auch verwenden. Die Wahrscheinlichkeit dafür ist jedoch extrem gering: GUI-Applikationen sind Remote nicht jedem Angreifer verfügbar; zudem sind GUI-Applikationen remote sehr langsam. Es gibt viele andere Editoren: kwrite, Nedit, Emacs, XEmacs, Xedit, vi, vim, gvim, ed, gedit, joe, elvis ...

Ein weiteres Beispiel wäre das folgende: Ein normaler Benutzer verwendet in der Regel keinen Compiler. Der Angreifer könnte Exploit-Code jedoch mit dem C-Compiler übersetzen und müsste alleine dafür schon diverse Programme aufrufen:

```
\$ gcc -o test test.c
Apr 26 14:39:22 projekt /bsd: fupids: new programm: 'gcc', uid: 1000
Apr 26 14:39:22 projekt /bsd: fupids: new programm: 'cpp0', uid: 1000
Apr 26 14:39:23 projekt /bsd: fupids: new programm: 'ccl1', uid: 1000
Apr 26 14:39:23 projekt /bsd: fupids: new programm: 'as', uid: 1000
Apr 26 14:39:24 projekt /bsd: fupids: new programm: 'collect2', uid: 1000
Apr 26 14:39:24 projekt /bsd: fupids: new programm: 'ld', uid: 1000
```

Man bedenke, dass dieses Programm zusätzlich noch selbst aufgerufen werden muss und daher wahrscheinlich noch einen Eintrag in der Logdatei erstellt und der Attacker-Level ansteigt.

Wichtig: Programme werden via I-Node Nummer ihrer Dateinamen identifiziert. Es ist also nicht möglich eine Datei mit dem Namen "ls" als Exploit zu tarnen.

FUPIDS ist zudem in der Lage vom Benutzer erzeugte, mögliche TCP- und UDP-Backdoors, die einen listen()-Syscall ausführen aufzuspüren und Netzwerkkarten, die in den promiscuous-Modus gesetzt wurden zu protokollieren. Letzteres Feature soll eventuelle Sniffing-Aktivitäten aufdecken. Der Administrator sollte in diesem Fall nach Programmnamen wie "dsniff" oder Ähnlichem ausschau halten. Hierzu später mehr im Protokollierungsformat.

## 5.1 Features

- Erstellung von Benutzerprofilen
- Überwachung der vom Benutzer aufgerufenen Programme
- Identifizierung der Programme über das Dateisystem auf Inode-Basis
- Protokollierung via syslogd (Remote-Protokollierung auf Logging-Server möglich)
- Überwachung von listen()-Syscalls der Benutzer
- Überwachung der Netzwerkschnittstellen auf von Benutzern eingeleitete promiscuous-Modi.
- FUPIDS ist erweiterbar
- FUPIDS ist transparent/unsichtbar für den Endanwender

## 5.2 Attacker-Level

Es gibt folgende Attacker-Level die im Bereich von 300 bis 999 liegen. Bei jeder Aktion wird – je nach Attacker-Level – ein verschieden großer Wert vom Attacker-Level abgezogen. Dies ist notwendig um zu verhindern, dass der Attacker-Level immer weiter steigt und nach einer gewissen Zeit jeder Benutzer als sehr gefährlicher Angreifer gilt.

- ab 300, 5 Punkte Abzug
- ab 400, 7 Punkte Abzug
- ab 700, 30 Punkte Abzug, "Low-Attacker-Level"
- ab 800, 50 Punkte Abzug, "Medium-Attacker-Level"
- ab 900, 80 Punkte Abzug, "High-Attacker-Level"
- ab 999, 99 Punkte Abzug, Filter für zu extreme Levelwerte

Diese Werte wurden so gesetzt, dass sie nach einigen Tests eine "gesunde" Anzahl an Logeinträgen ergaben. Der Administrator soll nicht das wichtigste verpassen, soll jedoch keinesfalls mit falschen, zu sensiblen, Logmeldungen überflutet werden.

### 5.2.1 Berechnung des Attacker-Levels

Die Berechnung wird in der Datei fupids.c in der Funktion

```
void upid_checkit(struct upid *np, struct mupid *mnp, int flag)
```

berechnet.

## 5.3 Protokollierungsformat

Die Protokollierung erfolgt auf der Rootkonsole und über den syslog-Daemon. Im Normalfall schreibt syslog die Logdaten dann in die Datei /var/log/messages. Die Datensätze von fupids können auf einfache Art und Weise mit dem grep-Befehl herausgefiltert werden:

```
projekt# grep fupids /var/log/messages
Apr 26 14:39:06 projekt /bsd: fupids: new user, uid 1000.
Apr 26 14:39:06 projekt /bsd: fupids: new programm: 'ksh', uid: 1000
Apr 26 14:39:10 projekt /bsd: fupids: new programm: 'ls', uid: 1000
Apr 26 14:39:15 projekt /bsd: fupids: new programm: 'more', uid: 1000
Apr 26 14:39:22 projekt /bsd: fupids: new programm: 'gcc', uid: 1000
Apr 26 14:39:22 projekt /bsd: fupids: new programm: 'cpp0', uid: 1000
Apr 26 14:39:22 projekt /bsd: fupids: new programm: 'cc1', uid: 1000
Apr 26 14:39:23 projekt /bsd: fupids: new programm: 'as', uid: 1000
Apr 26 14:39:23 projekt /bsd: fupids: new programm: 'collect2', uid: 1000
Apr 26 14:39:24 projekt /bsd: fupids: new programm: 'ld', uid: 1000
May  1 03:30:01 projekt /bsd: fupids: new user, uid 32767.
May  1 03:30:01 projekt /bsd: fupids: new programm: 'csh', uid: 32767
May  1 03:30:01 projekt /bsd: fupids: new programm: 'sh', uid: 32767
May  1 03:30:01 projekt /bsd: fupids: new programm: 'sed', uid: 32767
```

```

May  1 03:30:01 projekt /bsd: fupids: new programm: 'mktemp', uid: 32767
May  1 03:30:01 projekt /bsd: fupids: new programm: 'find', uid: 32767
May  1 03:30:01 projekt /bsd: fupids: new programm: 'sort', uid: 32767
May  1 03:31:38 projekt /bsd: fupids: new programm: 'locate.bigram', uid: 32767
May  1 03:31:39 projekt /bsd: fupids: new programm: 'awk', uid: 32767
May  1 03:31:39 projekt /bsd: fupids: new programm: 'locate.code', uid: 32767
May  1 03:31:40 projekt /bsd: fupids: new programm: 'rm', uid: 32767
May  1 03:31:40 projekt /bsd: fupids: new programm: 'cat', uid: 32767

```

### 5.3.1 Neue Programme

Wird ein neues Programm aufgerufen, erscheint, wie obig zu sehen ist, ein Eintrag der Form

```

Monat Tag Uhrzeit Hostname /Kernel: fupids: new programm:
      'Programm', uid: User-ID

```

### 5.3.2 promiscuous Modus

Beim Aufruf eines Programmes, welches eine Netzwerkkarte in den promiscuous Modus setzt, erscheint die Protokollausgabe wiederum in folgender Form:

```

Monat Tag Uhrzeit Hostname /Kernel: fupids: Schnittstelle in prromiscuous mode!
[proc: Programm pid: Prozess-ID | parent: Parent-Prozess ppid: Parent-PID |
uid: User-ID]

```

Beispielsweise:

```

May  4 12:22:17 projekt /bsd: fupids: sis0: set in promiscuous mode!
[proc: tcpdump pid: 19159 | parent: ksh ppid: 27535 | uid: 1000]

```

### 5.3.3 Attacker-Level

Sicherheitswarnungen, die ausgegeben werden, wenn Benutzer in einem, der obig definierten Securitylevel geraten, werden folgendermaßen protokolliert:

```

Monat Tag Uhrzeit Hostname /Kernel: fupids: Warn-Level, uid User-ID,
      Attacker-Level!

```

Beispielsweise:

```

May  4 12:22:17 projekt /bsd: fupids: low security warning, uid 1000,741!

```

### 5.3.4 listen()-Sockets

Angreifer starten häufig TCP- bzw. UDP-Backdoors. Diese müssen jedoch einen listen()-Syscall ausführen um Verbindungen akzeptieren zu können. Stellt FUPIDS solch einen Syscall fest, wird dieser in folgender Form protokolliert:

```
Monat Tag Uhrzeit Hostname /Kernel: fupids: user USER-ID, prog 'Programm':
listen syscall
```

Beispiel:

```
May 4 12:22:17 projekt /bsd: fupids: user 1000, prog 'backdoor':
listen syscall
```

## 5.4 Beispielüberwachung

Im Folgenden wurde das Programm /usr/sbin/tcpdump mit GUID- und SUID-Rechten versetzt. Anschliessend wurde die Netzwerkkarte von einem Benutzer in den promiscuous-Modus gesetzt.

```
projekt# chmod +s /usr/sbin/tcpdump
projekt# chmod +g /usr/sbin/tcpdump
projekt# su steffen
\$ ifconfig -a
...
...
\$ tcpdump -i sis0
May 4 12:21:57 projekt /bsd: fupids: new program: 'ifconfig', uid: 1000
May 4 12:22:17 projekt /bsd: fupids: new program: 'tcpdump', uid: 1000
May 4 12:22:17 projekt /bsd: fupids: sis0: set in promiscuous mode!
[proc: tcpdump pid: 19159 | parent: ksh ppid: 27535 | uid: 1000]
May 4 12:22:17 projekt /bsd: fupids: low security warning, uid 1000,741!
```

FUPIDS hat detektiert, dass das Programm ifconfig und tcpdump vom Benutzer mit der User-ID 1000 aufgerufen wurde. Er hat die Schnittstelle sis0 in den promiscuous Modus gesetzt. Dies wurde mit dem Programm tcpdump (Prozess-ID 19159) erledigt. Der Benutzer hat dabei die Korn-Shell ("ksh") verwendet. FUPIDS protokollierte darauf einen "low" Attacker-Level (741).

## 5.5 Installation

FUPIDS baut auf den OpenBSD-Kernelquellen vom Dezember 2003 auf. "Leider" verändern die Entwickler die Kernelsourcen täglich, doch sollte es mit etwas Glück möglich sein, FUPIDS auch in aktuelle Kernelsourcen einzupatchen. <sup>2</sup>

<sup>2</sup>Ein Test mit den Quellen vom Januar 2004 gelang problemlos.

Um die Installationsgrundlage zu schaffen, benötigt man OpenBSD, Version 3.4 und die Kernelquellen. Diese sind entweder via FTP von ftp.openbsd.org oder via CVS von einem der CVS-Mirrors, die auf OpenBSD.org gelistet sind, zu beziehen.

Die Quellen liegen im .tgz-Format vor und sind in das Verzeichnis /sys zu entpacken. Existiert /sys nicht, sollte es als Link auf /usr/src/sys erstellt werden.

FUPIDS selbst ist ebenfalls in eine .tgz-Datei gepackt und sollte in /sys entpackt werden:

```
projekt# tar -xzf fupids.tgz
fupids
fupids/sys
fupids/sys/kern
fupids/sys/kern/fupids.c
fupids/sys/sys
fupids/sys/sys/fupids.h
fupids/INSTALL
fupids/p_nif
fupids/p_kuipdsoc
fupids/p_kexec
```

Darauf hin sollten zunächst die Kernelpatches installiert werden:

```
projekt# patch -p0 < fupids/p_kexec
...
projekt# patch -p0 < fupids/p_kupidsoc
...
projekt# patch -p0 < p_nif
...
```

Der Patch p\_kexec patched die Datei kern/kexec.c. Diese beinhaltet den Code um Binärdateien auszuführen. FUPIDS klinkt sich beim Ausführen eines Programmes ein. Der Patch p\_kupidsoc patcht die listen-socket Syscall-Routine, p\_nif den Syscall um ein Netzwerk-Interface in den promiscuous-Modus zu setzen.

Anschliessend muss der Kernel über die neuen Projektdateien informiert werden. Dazu muss in der Datei /sys/conf/files die Datei fupids.c eingetragen werden.

```
projekt# vi /sys/conf/files
...
```

Die Dateien in fupids/sys und fupids/kern müssen nun noch in die Verzeichnisse /sys/sys und /sys/kern kopiert werden.

```
projekt# cp fupids/sys/* /sys/sys/
projekt# cp fupids/kern/* /sys/kern/
```

Jetzt muss nur noch die Kernelkonfiguration umgeschrieben und der Kernel neu übersetzt und installiert werden. Dazu erstellt man eine Kopie der GENERIC Konfigurationsdatei aus `/usr/src/sys/arch/i386/conf` und definiert in dieser Kopie die Option FUPIDS. Bei der Kompilierung wird diese Option später in ein C-Define umgewandelt.

```
projekt# cd /usr/src/sys/arch/i386/conf
projekt# cp GENERIC FUPIDS
projekt# vi FUPIDS
--- Nun folgende Zeile einfuegen:
option      FUPIDS
...
```

Nun muss die Konfiguration neu erstellt werden und der Kernel übersetzt werden:

```
projekt# config FUPIDS
projekt# cd ../compile/FUPIDS
projekt# make depend
...
...
projekt# make
...
...
mkdir -p /sys/arch/i386/compile/PROJECT/lib/kern
making sure the kern library is up to date...
'libkern.o' is up to date.
making sure the compat library is up to date...
'libcompat.a' is up to date.
sh /usr/src/sys/arch/i386/compile/PROJECT/../../../../conf/newvers.sh
cc -Werror -Wall -Wstrict-prototypes -Wmissing-prototypes
-Wno-uninitialized -Wno-format -Wno-main -O2 -nostdinc -I.
-I/usr/src/sys/arch/i386/compile/PROJECT/../../../../arch
-I/usr/src/sys/arch/i386/compile/PROJECT/../../../../DDDB
-DDIAGNOSTIC -DKTRACE -DKMEMSTATS -DPTRACE -DCRYPTO
-DSYSVMSG -DSYSVSEM -DSYSVSHM -DUVM_SWAP_ENCRYPT -DCOMPAT_25
-DCOMPAT_43 -DLKM -DFFS -DFFS_SOFTUPDATES -DQUOTA -DEXT2FS
-DMFS -DTCP_SACK -DTCP_ECN -DNFSCLIENT -DNFSSERVER -DCD9660
-DMSDOSFS -DFDESC -DFIFO -DKERNFS -DPORTAL -DPROCFS -DNULLFS
-DUMAPFS -DUNION -DINET -DALQT -DINET6 -DIPSEC -DPPP_BSDCOMP
-DPPP_DEFLATE -DBOOT_CONFIG -DI386_CPU -DI586_CPU
-DGPL_MATH_EMULATE -DUSER_PCICONF -DUSER_LDT -DAPERTURE
-DCOMPAT_LINUX -DCOMPAT_AOUT -DFUPIDS -DPCIVERBOSE
-DEISAVERVERBOSE -DUSBERVERBOSE -DWSDISPLAY_COMPAT_USL
-DWSDISPLAY_COMPAT_RAWKBD -DWSDISPLAY_DEFAULTSCREENS="6"
-DWSDISPLAY_COMPAT_PCVT -DPCIAGP -D_KERNEL -Di386 -c vers.c
rm -f bsd
ld -Ttext 0xd0100120 -e start -N -x -o bsd ${SYSTEM_OBJ} vers.o
text data bss dec hex filename
4295248 90868 838320 5224436 4fb7f4 bsd
```

```
projekt# cp ./bsd /bsd
```

Danach muss der Rechner rebootet werden. Um zu testen, ob FUPIDS läuft, sollte überprüft werden, ob der neue Kernel verwendet wird und ob der FUPIDS-Code aktiviert wurde. Letzteres bekommt man durch einen Benutzerlogin mit einer Benutzer-ID auf der Rootkonsole heraus.

Ersteres durch das uname-Programm:

```
projekt# uname -v
FUPIDS#1
```

## 5.6 Erweiterbarkeit

FUPIDS wurde so entwickelt, dass es eine "Einstiegs-Funktion" (`upid_userfind()`) bereit stellt. Diese kann in diverse Syscalls eingebaut werden. Zur Nutzung dieser Datei muss wiederum `sys/fupids.h` eingebunden werden. Etwa so:

```
#ifndef FUPIDS
#include <sys/fupids.h>
#endif
...
...
#ifdef FUPIDS
if(curproc->p_cred->p_ruid >= 1000)
upid_userfind(curproc->p_cred->p_ruid, NULL);
#endif
```

## 5.7 Datei-Inodes

Um die einzelnen, vom Benutzer gestarteten, Programme zu identifizieren, verwendet FUPIDS Inode-Nummern. Jede Datei im Dateisystem verfügt über eine eigene Inode-Nummer. Diese identifiziert den Inode-Eintrag der Datei. In diesem sind wiederum die Eigenschaften der Datei abgelegt.

FUPIDS kommt durch ein Macro namens VTOI (Vnode-To-Inode) aus `ufs/ufs/inode.h` an diese Inode-Nummer heran. Dabei wird der Vnode-Pointer des aktuellen Prozesses (`curproc`) übergeben:

```
#define UPID_ELEMENT (VTOI(curproc->p_textvp))->i_number%UPID_ARSIZE
```

## 5.8 Memory Allocation

Da im Kernel nicht ohne weiteres Speicher reserviert werden kann, muss FUPIDS sich darum kümmern, auch wirklich den benötigten Speicher zu bekommen. Im OpenBSD-Kernel stehen dazu zwei Möglichkeiten zur Verfügung: Entweder man verlangt unmittelbare Bereitstellung von Speicher, was zur Folge

haben kann, das momentan kein Speicher zur Verfügung steht und man keinen bekommt; oder man wartet, bis Speicher zur Verfügung steht und blockiert den Kernel.

FUPIDS benutzt `M_WAITOK` um sicher gehen zu können, dass auch wirklich der benötigte Speicher zugewiesen wird.

## 6 Glossar

### 6.1 Exploit

Ein Exploit ist ein Programm, das meistens im Quellcode vorliegt, um bekannte Schwachstellen von Programmen auszunutzen.

### 6.2 Kernelspace

Der Kernelspace ist ein gesicherter, für den Kernel reservierter, Speicherbereich. Anwendungen ist es nicht möglich auf diesen ohne weiteres (mit Ausnahme der Kernel-Virtual-Memory Library) zuzugreifen. Syscalls von Anwendungsprogrammen werden im Kernelspace ausgeführt.

### 6.3 Syscall

Ein Syscall ist eine Funktion, die vom Userspace aus aufgerufen wird und im Kernel ausgeführt wird. Hinter der Userspace-Funktion `printf(1)` versteckt sich bspw. der Syscall `write(2)`.

### 6.4 Userspace

Im Userspace werden Anwendungsprogramme ausgeführt.

## 7 Literatur

- The Design and Implementation of the 4.4 BSD Operating System; Marshall Kirk McKusick, Keith Bostic, Michael J. Karels, John S. Quarterman
- PANIC! UNIX System Crash Dump Analysis Handbook; Chris Drake, Kimberley A.D. Brown
- Linux - Unix Systemprogrammierung; Helmut Herold
- Advanced Programming in the UNIX Environment; Richard Stevens

- UNIX Internals: The New Frontiers; by Uresh Vahalia
- The Implementation (TCP/IP Illustrated, Volume 2) Gary R. Wright, W. Richard Stevens