Monitoring FAQ

Diese FAQ stammt aus Beiträgen vom Monitoring-Portal, wo die Themen „Icinga-Distribution für Mac OS X" und die Check_MK-Agenten für Mac OS X und OpenBSD vorkamen. Diese FAQ ist noch nicht vollständig, beantwortet aber viele spezielle Fragen zum Thema Monitoring mit Mac OS X.

Icinga-Distribution 1.7.X-X für Mac OS X Intel Systemvoraussetzungen

Läuft auf OS X 10.6 Snow Leopard, OS X 10.7 Lion und auf OS X 10.8 Mountain Lion.

Icinga-Distribution 1.7.X-X für Mac OS X PowerPC Systemvoraussetzungen

Erfordert mindestens einen Power Macintosh G3 und Mac OS X 10.3.9 oder höher. Bei mir läuft das Paket auf einem Power Mac G5, 2,0 GHz Dual mit 4 GB DDR SDRAM vom Jahr 2003 und auf einem Mac Mini G4, 1,42 GHz mit 1 GB DDR SDRAM vom Jahr 2005.

Ich habe es unter folgenden Systemen erfolgreich getestet:

Mac OS X 10.3.9 Panther PowerPC
Mac OS X 10.4.11 Tiger PowerPC
Mac OS X 10.5.8 Leopard PowerPC

Dank dem eingebauten PPC-Emulator Rosetta funktioniert diese Version auch unter:

Mac OS X 10.5.8 Leopard Intel
Mac OS X 10.6.8 Snow Leopard Intel
10.7.X Lion Intel und 10.8 Mountain Lion Intel enthalten keinen PPC-Emulator.

Leopard-Support (Icinga-Distribution 1.7.X-X für Mac OS X PowerPC)

Unter Leopard PowerPC läuft das Paket mit Hilfe der Bibliothek libltdl.3.dylib.

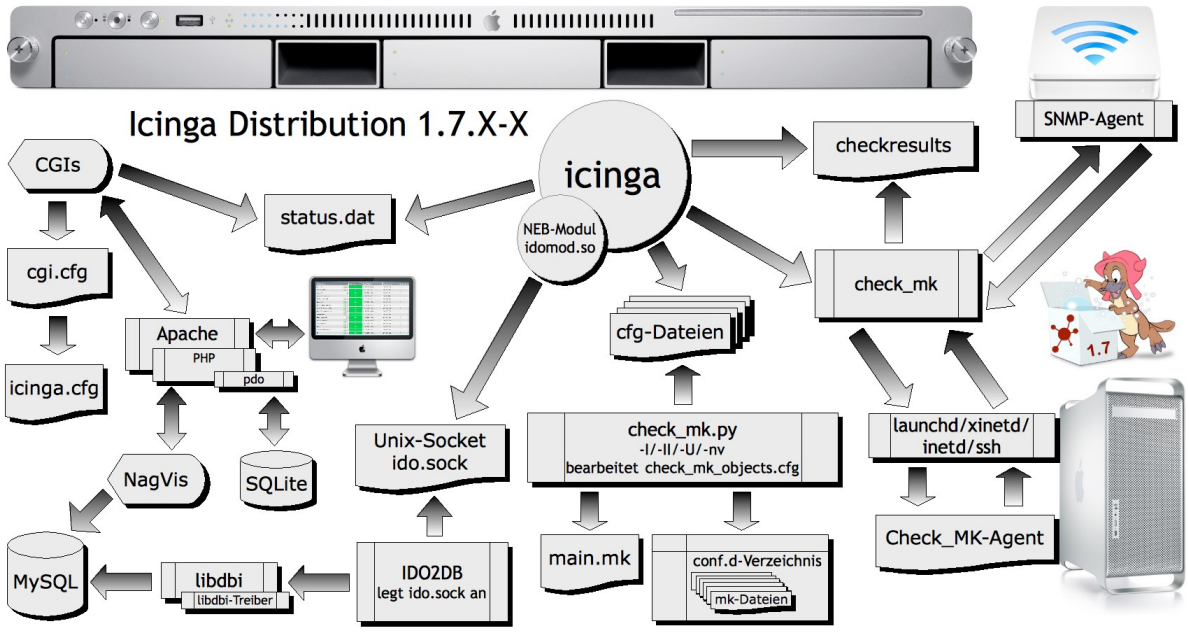
Das ist irgendwie keine schöne Lösung mit der libltdl.3.dylib. Ich kompiliere Icinga unter Mac OS X 10.4.11 PowerPC und habe dort mal geschaut, ob es eine statische Lib gibt. Und siehe da, es gibt die **libltdl.a**. Also bin ich schnell in das **base**-Verzeichnis gewechselt und nach einem **make clean** und **make** habe ich das Linker-Kommando vor mir gehabt. Dieses habe ich dann editiert:


```
gcc -g -O2 -I/Applications/Icinga_Start.app/Contents/Resources/include/
-DHAVE_CONFIG_H -DNSCORE -o icinga icinga.c broker.o nebmods.o
../common/shared.o checks.o config.o commands.o events.o flapping.o
logging.o macros-base.o netutils.o notifications.o sehandlers.o skiplist.o utils.o
profiler.o retention-base.o xretention-base.o comments-base.o xcomments-
base.o objects-base.o xobjects-base.o statusdata-base.o xstatusdata-base.o
perfdata-base.o xperfdata-base.o downtime-base.o xdowntime-base.o
-L/Applications/Icinga_Start.app/Contents/Resources/lib/  -lm  -lpthread
/Developer/SDKs/MacOSX10.3.9.sdk/usr/lib/libltdl.a
```

Kompilieren von IDOUtils unter Mac OS X 10.3.9 Panther

Der Compiler beschwert sich, dass suseconds_t nicht declariert ist. Auch das Einbinden der **types.h** hat nichts gebracht. Nun muß ich aber dazuschreiben, dass ich das Kompilieren unter dem alten 10.3.9 Panther-System versucht habe. Unter 10.4.11 Tiger gab es keine Probleme mit dem kompilieren der IDOUtils. Das meiste was unter Tiger kompiliert wurde, läuft auch unter Panther. Und das funktioniert auch in diesem Fall. Die IDOUtils, die ich unter Tiger kompiliert habe, funktionieren einwandfrei unter Panther. Danach habe ich diese, auch unter 10.5.8 Leopard einwandfrei testen können.

Aufbau der Icinga-Distribution für Mac OS X

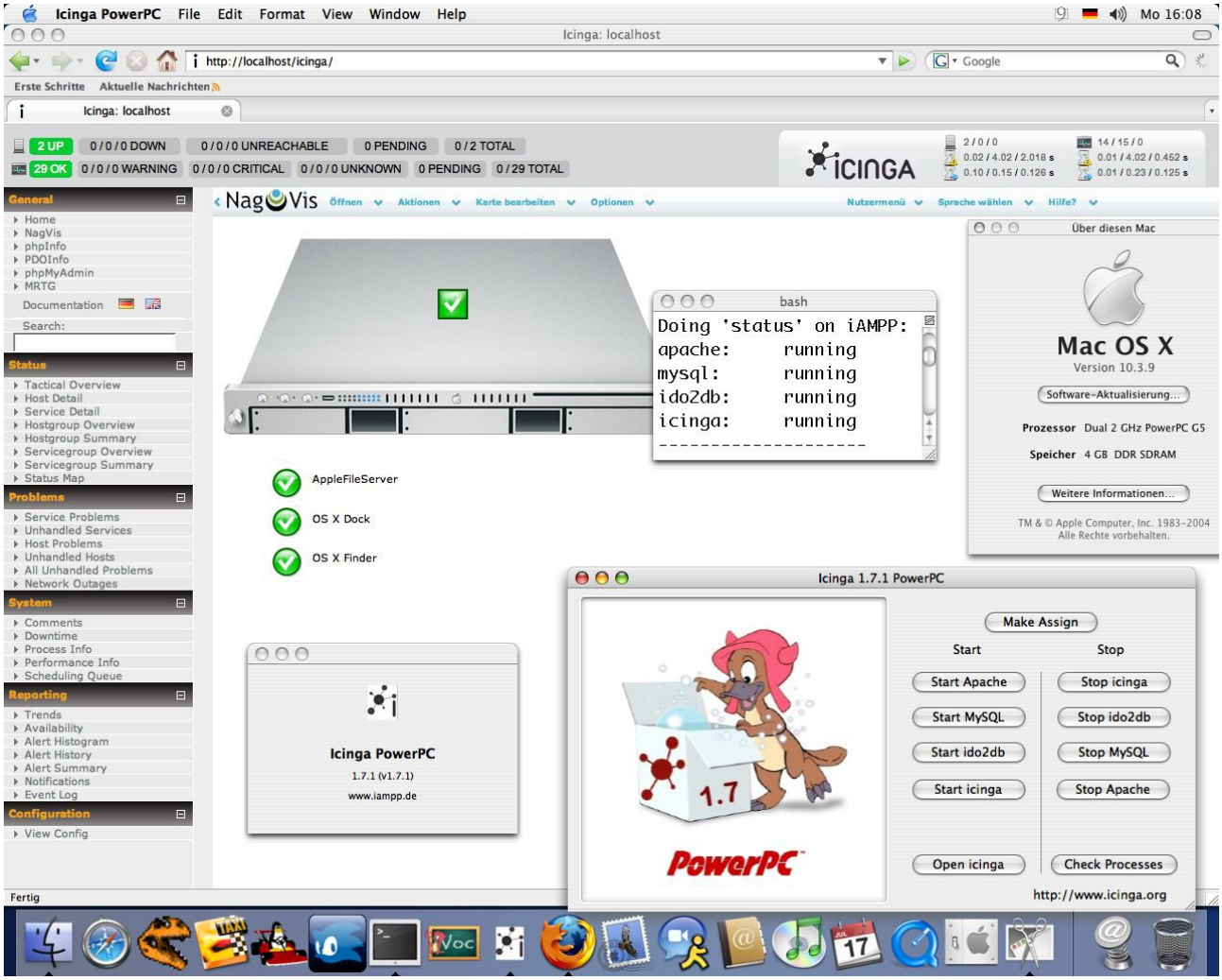


Hier eine komplette Liste der Programme, die in der Icinga-Distribution für Mac OS X PowerPC enthalten sind:

- M4 1.4.16
- Bison 2.5.1
- MySQL 5.4.1
- SQlite 3.3.6
- libXML 2.8.0
- PHP 5.2.17
- Python 2.7.3
- Apache 2.2.22
- phpMyAdmin 3.5.1
- icinga 1.7.1
- Nagios-Plugins 1.4.15
- libdbi 0.8.4
- libdbi-drivers 0.8.3.1
- gettext 0.18.1.1
- graphviz 2.28.0
- NagVis 1.6.6
- MRTG 2.17.4 mit check_mrtg
- Check_MK 1.2.0p2
- check_afs
- check_finder
- check_dock

Über einen Launcher kann die Icinga-Distri gesteuert werden.

Ich habe auf dem Launcher die Schaltfläche **Start ido2db** vor der **Start icinga**-Schaltfläche gelegt. Der Grund ist, dass ich öfter im Log gesehen habe, dass **IDOMOD.O** sich beschwert, dass die **ido.sock** noch nicht vorhanden ist. Das kommt davon, weil man naturgemäß von oben nach unten die Schaltflächen anklickt. Dadurch wird Icinga vor dem **IDO2DB** gestartet und das Modul **IDOMOD** ist schneller geladen als **IDO2DB** die **ido.sock** anlegen kann.



Verzeichnis conf.d bei den Icinga-Distributionen

Die Icinga-Distris haben noch kein **conf.d**-Verzeichnis. Dieses muß im gleichen Verzeichnis angelegt werden, wo die **main.mk** ist. Dadurch, dass alles was zur Distri gehört im App unter Programme ist, sieht der Pfad zum **conf.d**-Verzeichnis und der **main.mk** bei der Intel-Distri wie folgt aus:

/Applications/Icinga_1_7_1-2.app/icinga-data.app/Contents/Resources/etc/check_mk

Beispiel einer MK-Datei im conf.d-Verzeichnis

```
all_hosts = all_hosts + [

'christians-mac-mini',
'win2000',
]

ipaddresses.update({
  "christians-mac-mini" : "192.168.1.114",
  "win2000" : "192.168.1.107",
})

checks += [
  ( ["christians-mac-mini"], "ps", "AppleFileServer_Check",
( "~.*AppleFileServer", 1, 1, 1, 1 ) )
]

ignored_services += [
        ( [ "win2000" ],  [ "LOG Security" ] ),
        ( [ "win2000" ],  [ "LOG Application" ] ),
        ( [ "win2000" ],  [ "LOG System" ] ),
]
```

Netcat unter Mac OS X

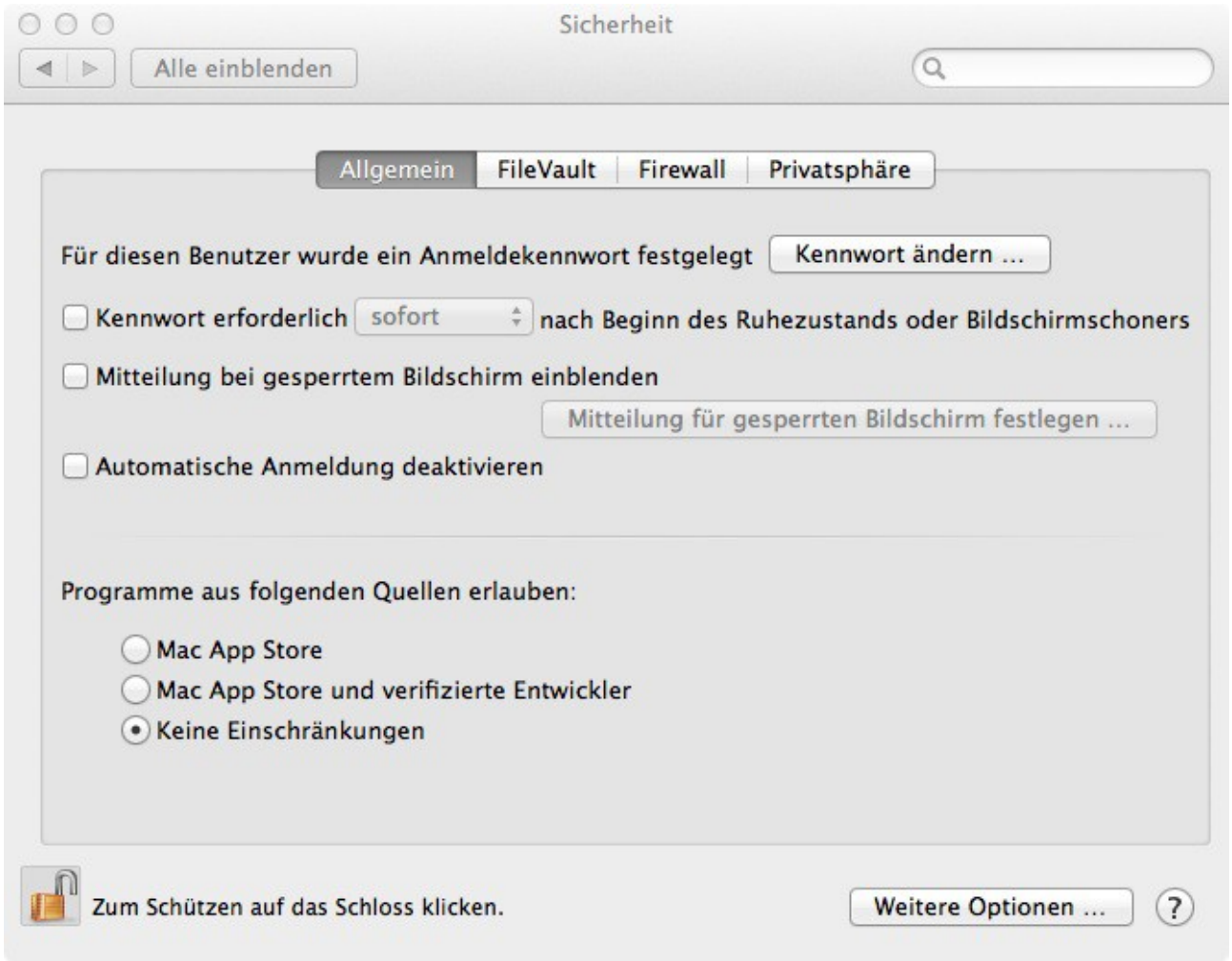
In der MK-Doku wird von **netcat** gesprochen, um die Ausgabe vom Agenten über das Netzwerk zu bekommen. Das Kommando **netcat** gibt es nicht unter OS X. Stattdessen könnt ihr **nc** verwenden.

```
nc <IP-Adresse> 6556
```

Icinga Distribution unter Mac OS X 10.8 Mountain Lion

Die **Icinga-Distri 1.7.1-2 (Intel)** funktioniert auch unter OS X 10.8 Mountain Lion. Eine wichtige Sache muß aber beachtet werden. Es geht um eine neue Funktion mit dem Namen Gatekeeper.

Gatekeeper soll vor allem unerfahrene Mac-Anwender vor gefährlichen oder mit Schadsoftware verseuchten Programmen schützen, indem, im günstigsten Fall, nur Programme aus dem Mac App Store installiert werden können. Doch auch andere Einstellungen sind möglich. Icinga ist nicht von Apple digital signiert und lässt sich mit den Standardeinstellungen nicht starten. Damit Icinga startet, muß unter den **Systemeinstellungen-Sicherheit** die Option **Keine Einschränkungen** im Bereich **Programme aus folgenden Quellen erlauben** aktiviert werden:



Der neue **Safari 6** hat scheinbar einen kleinen Bug. Wenn mit diesem das **Icinga CLASSIC-WEB** geöffnet wird, dann wird für jeden Frame das Zugangspasswort (htpasswd) verlangt. Meine Empfehlung ist, einfach einen anderen Webbrowser wie Firefox oder Chromium zu verwenden. Wenn Ihr unbedingt Safari 6 benutzen wollt, dann könnt ihr auch das Kennwort im

Schlüsselbund speichern.

Für alle die von Snow Leopard (**10.6.X**) auf 10.8 Mountain Lion umgestiegen sind und die **PowerPC-Distri** mit Hilfe von **Rosetta** verwendet haben, müssen jetzt die Intel-Distri verwenden, weil Apple, Rosetta schon seit 10.7 Lion nicht mehr unterstützt.

Mit der Icinga-Distri via SSH auf Mac OS X-Agents zugreifen

Ausgangsbasis:
Monitoring Server: Icinga Distribution 1.7.1-2 Intel unter OS X 10.8 Mountain Lion
Überwachter Host: Mac OS X 10.5.8 Leopard PowerPC mit Check_MK-Agent 1.2.0p2 für Mac OS X

Überwachter Host:

```
sudo rm /Library/LaunchDaemons/de.jvm.check_mk-agent.plist
```

Damit gibt Launchd am Port 6556 nicht mehr die Ausgabe vom Agenten aus.

Monitoring-Server:

```
ssh-keygen -t rsa
```

```
cd .ssh
```

```
scp id_rsa.pub christian@192.168.1.114:/Users/christian/.ssh
```

Monitoring FAQ

Überwachter Host:

```
cd .ssh
```

```
cat id_rsa.pub >> authorized_keys
```

Monitoring Server:

```
ssh christian@192.168.1.114
```

```
# Put your host names here
# all_hosts = [ 'localhost' ]
all_hosts = [ 'OpenBSD' , 'Leopard-Server' , 'Lubuntu' ]

ipaddresses = {
  "OpenBSD" : "192.168.1.128",
  "Leopard-Server" : "192.168.1.114",
  "Lubuntu" : "192.168.1.144",
}

inventory_check_interval = 120

linux_nic_check = "legacy"

datasource_programs = [
 ( "ssh -l christian <IP> check_mk_agent.openbsd", [ 'OpenBSD' ] ),
 ( "ssh -l christian <IP> check_mk", [ 'Leopard-Server' ] ),
 ( "ssh -l christian <IP> check_mk_agent", [ 'Lubuntu' ] ),
]
```

```
./check_mk.py -II Leopard-Server
```

```
cpu.loads 1 new checks
cpu.threads 1 new checks
df 3 new checks
mrpe 5 new checks
netctr.combined 4 new checks
ntp.time 1 new checks
tcp_conn_stats 1 new checks
uptime 1 new checks
```

```
./check_mk.py -nv Leopard-Server
```


```
CPU load OK - 15min load 0.03 at 1 CPUs
FS_ DISK OK - free space: / 23814 MB (61% inode=60%);
FS_var DISK OK - free space: / 23814 MB (61% inode=60%);
LOAD OK - load average: 0.00, 0.01, 0.03
NIC en0 counters OK - Receive: 0.00 MB/sec - Send: 0.00 MB/sec
NIC fw0 counters OK - Receive: 0.00 MB/sec - Send: 0.00 MB/sec
NIC gif0 counters OK - Receive: 0.00 MB/sec - Send: 0.00 MB/sec
NIC stf0 counters OK - Receive: 0.00 MB/sec - Send: 0.00 MB/sec
NTP Time OK - sys.peer - stratum 2, offset 3.27 ms, jitter 0.69 ms, last
reached 52 secs ago (synchronized on 17.72.255.11)
Number of threads OK - 51 threads
TCP Connections OK - ESTABLISHED: 1
Uptime OK - up since Mon Jul 30 16:00:18 2012 (0d 00:46:05)
dock Dock: Running\n
finder Finder: Running\n
fs_/ OK - 39.3% used (15.03 of 38.3 GB), (levels at 80.0/90.0%), trend:
0.00B / 24 hours
fs_/Volumes/Boot OK - 14.8% used (0.01 of 0.1 GB), (levels at 80.0/90.0%),
trend: 0.00B / 24 hours
fs_/Volumes/Panther2 OK - 40.8% used (15.56 of 38.2 GB), (levels at
80.0/90.0%), trend: 0.00B / 24 hours
```

```
./check_mk.py -U leopard-Server
```

```
Generating Nagios configuration...OK
Precompiling host checks...OK
Successfully created Nagios configuration file
/tmp/icinga.app/Contents/Resources/etc/objects/check_mk_objects.cfg.

Please make sure that file will be read by Nagios.
You need to restart Nagios in order to activate the changes.
```

Wichtig: Es gibt unter Mac OS X kein Verz. **/etc/init.d**. Das heißt, dass ein **./check_mk.py -R Leopard-Server** hier nicht funktioniert. Das App soll auch keine Veränderungen im OS X-System vornehmen. Deshalb ist es nicht geplant, die Option **-R** mit dem Start-Mechanismus (Launchd/plist-Datei) von Mac OS X zu verknüpfen.

Also stoppen und starten wir Icinga über den Launcher.

Danach ist der Leopard-Server im Icinga CLASSIC-WEB zu sehen.

Ein Beispiel einer zusätzlichen MK-Datei mit SSH-Zugriff auf einen Host:

```
all_hosts = all_hosts + [
        'Lubuntu',
        ]

ipaddresses.update ({
       "Lubuntu" : "192.168.1.144",
             })

datasource_programs = datasource_programs + [
       ("ssh -l christian <IP> check_mk_agent", [ 'Lubuntu' ]),
       ]
```

ssh-copy-id unter Mac OS X

Es gibt kein **ssh-copy-id** unter Mac OS X.

Download und Installation:

```
sudo curl "http://public.zettabyte.eu/ssh-copy-id" -o /usr/bin/ssh-copy-id
```

```
sudo chmod +x /usr/bin/ssh-copy-id
```

ssh-copy-id ist viel einfacher und läuft auch auf älteren Mac OS X-Systemen.

```
ssh-copy-id christian@192.168.1.144
```

SNMP

Nach der Abfrage eines Check_MK-Agenten über SSH, folgt jetzt der Versuch einen Windows 2000-Rechner via SNMP abzufragen.

Ausgangsbasis:
Monitoring Server: Icinga Distribution 1.7.1-7 PowerPC unter Mac OS X 10.3.9 Panther
Überwachter Host: Windows 2000 Professional SP4

Ihr seht, dass hier zwei Rentner in Kontakt treten. Was hier einfach toll ist, dass auf den alten Systemen Software von heute läuft. Der neueste Check_MK-Agent 1.2.0p2 funktioniert tadellos unter Win2k. Diesen habe ich aber für diesen Test deaktiviert.

OK, erstmal eine Abfrage laut MK-Doku von SNMP:

```
snmpget -v1 -c public 192.168.1.107 sysDescr.0
```

```
SNMPv2-MIB::sysDescr.0 = STRING: Hardware: x86 Family 6 Model 10 Stepping 5 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.0 (Build 2195 Uniprocessor Free)
```

xeno2.mk:

```
all_hosts = all_hosts + [ 'Windows 2000|snmp' ]

ipaddresses.update ({
                "Windows 2000" : "192.168.1.107"
                })
```

```
./check_mk.py -II "Windows 2000"
```

```
hr_cpu            1 new checks
hr_fs             1 new checks
hr_mem            1 new checks
if                1 new checks
snmp_info         1 new checks
snmp_uptime       1 new checks
```

OK, alles wird im CLASSIC WEB angezeigt.

Check_MK-Agent für OpenBSD

OpenBSD arbeitet mit **Inetd**. Einrichtung des Agenten:

```
cp check_mk_agent.openbsd /usr/bin
```


```
vi /etc/inetd.conf
```

inetd.conf:

```
check_mk    stream    tcp    nowait    root    /usr/bin/check_mk_agent.openbsd
```

```
vi /etc/services
```

services:

```
check_mk  6556/tcp
```

OK, MRPE brauche ich auch noch. Nagios-Plugins installieren:

```
export
PKG_PATH=ftp://openbsd.cs.fau.de/pub/OpenBSD/5.0/packages/`machine
-a`/
```

```
pkg_add -v nagios-plugins
```

```
nagios-plugins-1.4.15p1: ok
Look in /usr/local/share/doc/pkg-readmes for extra documentation.
--- +nagios-plugins-1.4.15p1 -------------------
The check_dhcp and check_icmp plugins need to run with superuser
privileges. For security reasons they are not installed suid root
by default. If you want to use them, you have to either change
their mode manually or use systrace's privilege elevation feature.
```

mrpe.cfg erstellen:

```
vi /etc/mrpe.cfg
```

```
LOAD      /usr/local/libexec/nagios/check_load -w 2 -c 5
FS_var    /usr/local/libexec/nagios/check_disk /var
FS_/    /usr/local/libexec/nagios/check_disk /
FS_usr    /usr/local/libexec/nagios/check_disk /usr
FS_home    /usr/local/libexec/nagios/check_disk /home
FS_tmp    /usr/local/libexec/nagios/check_disk /tmp
FS_usr/X11R6 /usr/local/libexec/nagios/check_disk /usr/X11R6
FS_usr/local /usr/local/libexec/nagios/check_disk /usr/local
FS_usr/src /usr/local/libexec/nagios/check_disk /usr/src
FS_usr/obj /usr/local/libexec/nagios/check_disk /usr/obj
```

OK, dann mal schauen, ob MRPE funktioniert:

```
nc 192.168.1.128 6556
```

```
<<<mrpe>>>
(check_load) LOAD 0 OK - load average: 0.10, 0.18, 0.16|
load1=0.101;2.000;5.000;0; load5=0.182;2.000;5.000;0;
load15=0.161;2.000;5.000;0;
(check_disk) FS_var 0 DISK OK - free space: /var 4150 MB (99%
inode=99%);| /var=28MB;;;0;4398
(check_disk) FS_/ 0 DISK OK - free space: / 849 MB (89% inode=98%);|
/=104MB;;;0;1004
(check_disk) FS_usr 0 DISK OK - free space: /usr 1547 MB (80%
inode=95%);| /usr=364MB;;;0;2012
(check_disk) FS_home 0 DISK OK - free space: /home 17408 MB (99%
inode=99%);| /home=0MB;;;0;18325
(check_disk) FS_tmp 0 DISK OK - free space: /tmp 2639 MB (99%
inode=99%);| /tmp=0MB;;;0;2778
(check_disk) FS_usr/X11R6 0 DISK OK - free space: /usr/X11R6 788 MB (82%
inode=93%);| /usr/X11R6=165MB;;;0;1004
(check_disk) FS_usr/local 0 DISK OK - free space: /usr/local 3988 MB (78%
inode=93%);| /usr/local=1084MB;;;0;5340
(check_disk) FS_usr/src 0 DISK OK - free space: /usr/src 1587 MB (99%
inode=99%);| /usr/src=0MB;;;0;1670
(check_disk) FS_usr/obj 0 DISK OK - free space: /usr/obj 1912 MB (99%
inode=99%);| /usr/obj=0MB;;;0;2012
```

OK, dann kommt Check_MK ins Spiel.

openbsd.mk:

```
all_hosts = all_hosts + [
'OpenBSD',
]

ipaddresses.update({
  "OpenBSD" : "192.168.1.128",
})
```


```
./check_mk.py -I OpenBSD
```

```
cpu.loads          1 new checks
cpu.threads        1 new checks
mrpe              10 new checks
uptime             1 new checks
```

```
./check_mk.py -nv OpenBSD
```

```
Check_mk version 1.2.0p2
CPU load             OK - 15min load 0.15 at 1 CPUs
FS_/                DISK OK - free space: / 849 MB (89% inode=98%);
FS_home              DISK OK - free space: /home 17408 MB (99% inode=99%);
FS_tmp              DISK OK - free space: /tmp 2639 MB (99% inode=99%);
FS_usr              DISK OK - free space: /usr 1547 MB (80% inode=95%);
FS_usr/X11R6         DISK OK - free space: /usr/X11R6 788 MB (82%
inode=93%);
FS_usr/local         DISK OK - free space: /usr/local 3988 MB (78%
inode=93%);
FS_usr/obj           DISK OK - free space: /usr/obj 1912 MB (99% inode=99%);
FS_usr/src          DISK OK - free space: /usr/src 1587 MB (99% inode=99%);
FS_var              DISK OK - free space: /var 4150 MB (99% inode=99%);
LOAD                OK - load average: 0.20, 0.16, 0.15
Number of threads    OK - 42 threads
Uptime              OK - up since Sat Jul 21 17:48:49 2012 (0d 01:05:32)
OK - Agent version 1.2.0p2, execution time 0.2 sec|execution_time=0.184
```

```
./check_mk.py -U OpenBSD
```

```
Generating Nagios configuration...OK
Precompiling host checks...OK
Successfully created Nagios configuration file
/tmp/icinga.app/Contents/Resources/etc/objects/check_mk_objects.cfg.

Please make sure that file will be read by Nagios.
You need to restart Nagios in order to activate the changes.
```

Check_MK erwartet eine **df**-Ausgabe mit Angabe des Dateisystems. Leider hat das **df** von OpenBSD nicht die Option **-T** die diese Ausgabe ermöglicht. Wenn ich die Option **-T** weglasse, dann wird die Ausgabe von Check_MK nicht verarbeitet. OK, dann muß ich ein wenig schummeln wie beim OS X-Agenten:

```
df -kPt ffs | sed -e 's/^\([^ ][^ ]*\) \(.*\)$/\1 ffs \2/' | sed 1d
```

<<<netctr>>> aktivieren

main.mk

```
linux_nic_check = "legacy"
```

<<<netctr>>> -Problem im OpenBSD-Agenten

Das Problem ist, dass die Werte Bytes in, Packets in usw. unter Linux aus dem **Proc**-Verzeichnis gewonnen werden (**/proc/net/dev**) und OpenBSD kein **/proc**-Dateisystem besitzt. Check_MK benötigt eine ganz bestimmte Ausgabe, die Bytes, Packets, Errors und Colls gleichzeitig enthält. Unter OpenBSD gibt es die Möglichkeit, mit Hilfe von **netstat**, die Ausgabe von der Linux-Ausgabe mit den geforderten Werten nachzubauen. Die Option **-b** zeigt die Bytes-Ausgabe und ohne **-b** die Packets-Ausgabe. Ich kann nicht beides gleichzeitig in einer Zeile ausgeben. Also musste ich einen sehr komplizierten Weg einschlagen, um genau die richtige Ausgabe mit den richtigen Werten zu erhalten, die Check_MK verarbeiten kann.

Hier der neue OpenBSD-Agent:

```
#!/bin/sh

# +------------------------------------------------------------------+
# |             ____ _               _        __  __ _  __           |
# |            / ___| |__   ___  ___| | __   |  \/  | |/ /           |
# |           | |   | '_ \ / _ \/ __| |/ /   | |\/| | ' /            |
# |           | |___| | | |  __/ (__|   <    | |  | | . \            |
# |            \____|_| |_|\___|\___|_|\_\___|_|  |_|_|\_\           |
# |                                                                  |
# | Copyright Mathias Kettner 2012             mk@mathias-kettner.de |
# +------------------------------------------------------------------+
#
# This file is part of Check_MK.
# The official homepage is at http://mathias-kettner.de/check_mk.
#
# check_mk is free software;  you can redistribute it and/or modify it
# under the  terms of the  GNU General Public License  as published by
# the Free Software Foundation in version 2.  check_mk is  distributed
# in the hope that it will be useful, but WITHOUT ANY WARRANTY;  with-
# out even the implied warranty of  MERCHANTABILITY  or  FITNESS FOR A
# PARTICULAR PURPOSE. See the  GNU General Public License for more de-
# ails.  You should have  received  a copy of the  GNU  General Public
# License along with GNU Make; see the file  COPYING.  If  not,  write
# to the Free Software Foundation, Inc., 51 Franklin St,  Fifth Floor,
# Boston, MA 02110-1301 USA.

# Author: Lars Michelsen <lm@mathias-kettner.de>
#         Florian Heigl <florian.heigl@gmail.com>

# NOTE: This agent has been adapted from the Check_MK FreeBSD agent.

# Remove locale settings to eliminate localized outputs where possible
export LC_ALL=C
unset LANG

export MK_LIBDIR="/usr/lib/check_mk_agent"
export MK_CONFDIR="/etc"

# Make sure, locally installed binaries are found
PATH=$PATH:/usr/local/bin

# All executables in PLUGINSDIR will simply be executed and their
# ouput appended to the output of the agent. Plugins define their own
# sections and must output headers with '<<<' and '>>>'
PLUGINSDIR=$MK_LIBDIR/plugins

# All executables in LOCALDIR will by executabled and their
# output inserted into the section <<<local>>>. Please refer
# to online documentation for details.
```


```
LOCALDIR=$MK_LIBDIR/local


# close standard input (for security reasons) and stderr
if [ "$1" = -d ]
then
    set -xv
else
    exec <&- 2>/dev/null
fi

echo '<<<check_mk>>>'
echo Version: 1.2.0p2
echo AgentOS: openbsd

osver="$(uname -r)"

echo '<<<df>>>'
df -kPt ffs | sed -e 's/^\([^ ][^ ]*\) \(.*\)$/\1 ffs \2/' | sed 1d

# processes including username, without kernel processes
echo '<<<ps>>>'
COLUMNS=10000
ps ax -o user,vsz,rss,pcpu,command | sed -e 1d -e 's/ *\([^ ]*\) *\([^ ]
*\) *\([^ ]*\) *\([^ ]*\) */(\1,\2,\3,\4) /'

echo '<<<cpu>>>'
echo `sysctl -n vm.loadavg | tr -d '{}'` `top -b -n 1 | grep -E '^[0-9]+
processes' | awk '{print $3"/"$1}'` `sysctl -n hw.ncpu`

echo '<<<uptime>>>'
echo `date +%s` - `sysctl -n kern.boottime | cut -d' ' -f 4,7 | tr ','
'.' | tr -d ' '` | bc

echo '<<<netctr>>>'
# BI= Bytes in
# PI= Packets in
# EI= Errors in
# EO= Errors out
# BO= Bytes out
# PO= Packets out
# CO= Colls

Z1=1
Z2=p

date +%s
while [ $Z1 -lt 15 ]
  do
  BI=$( netstat -inb | egrep -v Name | grep Link | awk '{print $1" "$5}'
```


```
| sed -ne $Z1$Z2 )
  PI=$( netstat -in | egrep -v Name | grep Link | awk '{print $5}' | sed
-ne $Z1$Z2 )
  EI=$( netstat -in | egrep -v Name | grep Link | awk '{print $6}' | sed
-ne $Z1$Z2 )
  FF1="0 0 0 0 0"
  BO=$( netstat -inb | egrep -v Name | grep Link | awk '{print $6}' | sed
-ne $Z1$Z2 )
  PO=$( netstat -in | egrep -v Name | grep Link | awk '{print $7}' | sed
-ne $Z1$Z2 )
  EO=$( netstat -in | egrep -v Name | grep Link | awk '{print $8}' | sed
-ne $Z1$Z2 )
  CO=$( netstat -in | egrep -v Name | grep Link | awk '{print $9}' | sed
-ne $Z1$Z2 )
  FF2="0 0"
    if [ "$PI" -gt "0" ]
      then
        echo $BI $PI $EI $FF1 $BO $PO $EO $FF2 $CO $FF2
    fi
  Z1=$(($Z1+1))
done

# IPMI-Data (Fans, CPU, temperature, etc)
# needs the sysutils/ipmitool and kldload ipmi.ko
if which ipmitool >/dev/null ; then
    echo '<<<ipmi>>>'
    ipmitool sensor list \
        | grep -v 'command failed' \
        | sed -e 's/ *| */|/g' -e "s/ /_/g" -e 's/_*$//' -e 's/|/ /g' \
        | egrep -v '^[^ ]+ na ' \
        | grep -v ' discrete '
fi

if which mailq >/dev/null 2>&1 && getent passwd postfix >/dev/null 2>&1;
then
  echo '<<<postfix_mailq>>>'
  mailq | tail -n 6
fi


# Einbinden von lokalen Plugins, die eine eigene Sektion ausgeben
if cd $PLUGINSDIR
then
  for skript in $(ls)
  do
    if [ -x "$skript" ] ; then
        ./$skript
    fi
  done
fi
```


```
# Lokale Einzelchecks
echo '<<<local>>>'
if cd $LOCALDIR
then
  for skript in $(ls)
  do
    if [ -x "$skript" ] ; then
        ./$skript
    fi
  done
fi

# MK's Remote Plugin Executor
if [ -e "$MK_CONFDIR/mrpe.cfg" ]
then
    echo '<<<mrpe>>>'
    grep -Ev '^[[:space:]]*($|#)' "$MK_CONFDIR/mrpe.cfg" | \
    while read descr cmdline
    do
        PLUGIN=${cmdline%% *}
        OUTPUT=$(eval "$cmdline")
        echo -n "(${PLUGIN##*/}) $descr $? $OUTPUT" | tr \\n \\1
        echo
    done
fi
```

Check_MK-Agent für Mac OS X

Achtung: Alle OS X-Versionen ab 10.4.0 Tiger verwenden **Launchd**.

Kleine Anleitung für **Mac OS X 10.4.0 Tiger oder höher**:

1. Im Pfad **/opt/omd/versions/<Versions-Nr.>/share/check_mk/agents** deiner OMD-Installation, findest du das Shellscript **check_mk_agent.macosx**. Das ist der Mac OS X-Agent.
2. Kopiere dieses Shellscript via scp, ftp usw. auf deinen überwachten OS X-Rechner. Am besten gleich nach **/usr/bin/**. Benenne dieses dann in **check_mk** um.
3. Launchd-Startskript erstellen:


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com DTDs/PropertyList-1.0.dtd">
   <plist version="1.0">
  <dict>
  <key>KeepAlive</key>
  <dict>
  <key>NetworkState</key>
  <true/>
  </dict>
  <key>UserName</key>
  <string>root</string>
  <key>GroupName</key>
  <string>wheel</string>
  <key>ProgramArguments</key>
  <array>
  <string>/usr/bin/check_mk</string>
  </array>
  <key>Sockets</key>
  <dict>
  <key>Listeners</key>
  <dict>
  <key>SockServiceName</key>
  <string>6556</string>
  <key>SockType</key>
  <string>stream</string>
  <key>SockFamily</key>
  <string>IPv4</string>
  </dict>
  </dict>
  <key>inetdCompatibility</key>
  <dict>
  <key>Wait</key>
  <false/>
  </dict>
  <key>Label</key>
  <string>de.jvm.check_mk-agent</string>
  </dict>
</plist>
```

Diese Datei muß dann nach **/Library/LaunchDaemons** kopiert werden. OS X rebooten. Danach kann der Agent am **Port 6556** abgefragt werden.

Mac OS X 10.3.X Panther und **Mac OS X 10.2.X Jaguar**:

1. Im Pfad **/opt/omd/versions/<Versions-Nr.>/share/check_mk/agents** deiner OMD-Installation findest du das Shellscript **check_mk_agent.macosx**.
2. Kopiere dieses Shellscript via scp, ftp usw. auf deinen überwachten OS X-Rechner. Am besten gleich nach **/usr/bin/**. Benenne dieses dann in **check_mk_agent** um.
3. Xinetd-Datei erstellen: **vi /etc/xinetd.d/check_mk_agent**

```
service check_mk
{
        type            = UNLISTED
        port            = 6556
        socket_type     = stream
        protocol        = tcp
        wait            = no
        user            = root
        server          = /usr/bin/check_mk_agent
# configure the IP address(es) of your Nagios server here:
#       only_from       = 192.168.1.106
        disable         = no
}
```

Xinetd restarten:

```
/System/Library/StartupItems/IPServices/IPServices start
```

Danach kann der Agent am **Port 6556** abgefragt werden.

Xinetd wurde mit OS X 10.2 eingeführt:
http://support.apple.com/kb/TA20863?viewlocale=en_US.

Für Mac OS X 10.1 muss **Inetd** konfiguriert werden.

Das sind Variablen, die zusätzliche Scripte und Konfigurationen einbinden können:

```
export MK_LIBDIR="/to/be/changed"
export MK_CONFDIR="/to/be/changed"
```

Beispiel OpenBSD-Agent (verkürzte Ausgabe):

```
export MK_LIBDIR="/usr/lib/check_mk_agent"
export MK_CONFDIR="/etc"

PLUGINSDIR=$MK_LIBDIR/plugins

LOCALDIR=$MK_LIBDIR/local

# Einbinden von lokalen Plugins, die eine eigene Sektion ausgeben
if cd $PLUGINSDIR
then
  for skript in $(ls)
  do
    if [ -x "$skript" ] ; then
        ./$skript
    fi
  done
fi

# Lokale Einzelchecks
echo '<<<local>>>'
if cd $LOCALDIR
then
  for skript in $(ls)
  do
    if [ -x "$skript" ] ; then
        ./$skript
    fi
  done
fi

# MK's Remote Plugin Executor
if [ -e "$MK_CONFDIR/mrpe.cfg" ]
then
    echo '<<<mrpe>>>'
    grep -Ev '^[[:space:]]*($|#)' "$MK_CONFDIR/mrpe.cfg" | \
    while read descr cmdline
    do
```


```
        PLUGIN=${cmdline%% *}
        OUTPUT=$(eval "$cmdline")
        echo -n "(${PLUGIN##*/}) $descr $? $OUTPUT" | tr \\n \\1
        echo
    done
fi
```

Ich überwache 3 Services, die nicht vom Agenten abgefragt werden, über die **mrpe.cfg**. Ich musste bei mrpe, dass **-n** bei **echo** entfernen. Weil als Ausgabe im Monitoring **-n** angezeigt wurde mit einem UNKNOWN.
Ansonsten funktioniert MRPE super auf OS X. Ich habe einmal mit der Icinga-Distribution 1.7 für OS X und einmal mit OMD 0.54 die MRPE-Checks inventarisiert. Überwachter OS X-Host: **/etc/mrpe.cfg**

```
finder  /usr/lib/nagios/plugins/check_finder
dock    /usr/lib/nagios/plugins/check_dock
applefileserver    /usr/lib/nagios/plugins/check_afs
```

/usr/lib/nagios/plugins/check_afs

```
#!/bin/bash

ps -A | grep AppleFileServer | grep -v "grep AppleFileServer" | grep -v
"check_afs" > /dev/null

if [ $? -ne 0 ]
then
echo "AppleFileServer: Not Running"
exit 2
else
echo "AppleFileServer: Running"
exit 0
fi
```

Noch eine Möglichkeit ist, diesen Prozess direkt in einer mk-Datei abzufragen:

```
checks += [
  ( ["christians-mac-mini"], "ps", "AppleFileServer_Check",
( "~.*AppleFileServer", 1, 1, 1, 1 ) )
]
```

Von der Icinga-OSX-Distribution 1.7.X-X aus, den Agenten abfragen:

```
./check_mk.py -II 192.168.1.114
```

```
cpu.loads          1 new checks
cpu.threads        1 new checks
df                 3 new checks
uptime             1 new checks
```

```
./check_mk.py --debug 192.168.1.114
```

```
Reading default settings from ./defaults
Reading config file
/tmp/icinga.app/Contents/Resources/etc/check_mk/main.mk...
Check_mk version 1.1.12p7
CPU load              OK - 15min Load 0.08 at 1 CPUs
Number of threads     OK - 45 threads
Uptime                OK - up since Thu May 17 22:14:20 2012 (0d 00:48:30)
fs_/                  OK - 35.4% used (13.55 of 38.3 GB), (levels at 80.0/90.0%)
fs_/Volumes/Boot      OK - 5.9% used (0.00 of 0.1 GB), (levels at 80.0/90.0%)
fs_/Volumes/Panther2 OK - 32.9% used (12.57 of 38.2 GB), (levels at
80.0/90.0%)
OK - Agent version 1.1.12p7, execution time 0.2 sec|execution_time=0.230
```

Ich kann also den Agenten ohne Probleme erreichen. Dann habe ich diesen in die **main.mk** eingetragen:

```
# Put your host names here
# all_hosts = [ 'localhost' ]
all_hosts = [ 'christian-virtual-machine' , 'christians-mac-mini' ]

ipaddresses = {
  "christian-virtual-machine" : "192.168.1.143",
  "christians-mac-mini" : "192.168.1.114",
}
```

Dann das Update vom Config-File durchgeführt:

```
./check_mk.py -II --update 192.168.1.114
```


```
Generating Nagios configuration...OK

Precompiling host checks...OK
Successfully created Nagios configuration file
/tmp/icinga.app/Contents/Resources/etc/objects/check_mk_objects.cfg.
```

omd-0.55.20120806: Source install (PPC64)

Ich habe unter **Lubuntu 12.04 PowerPC64** eine Source-Installation durchgeführt. Das Source-Paket habe ich von: **http://nightly.omdistro.org/2012-08-06/src/**. Dort habe ich das Paket "**omd-0.55.20120806-src.tar.gz**" heruntergeladen.

```
make version
```

Version setzen:

```
0.55.20120806
```

Installation:

```
make && make pack && make setup
```

Abhängigkeiten auflösen:

```
omd setup
```

OMD-Gruppe hinzufügen:

```
groupadd omd
```

DEB-Paket erstellen:

```
make deb
```

Monitoring FAQ

Site erstellen:

```
omd create xenosite
```

OMD starten (alle sites):

```
omd start
```

OMD-Status prüfen:

```
omd status
```

Nach der Installation einer neuen OMD:

```
omd update xenosite
```

OMD-Site stoppen:

```
omd stop xenosite
```

Xinetd für livestatus konfigurieren:

```
omd config set LIVESTATUS_TCP on
```

Alle installierten OMD-Versionen anzeigen:

```
omd versions
```

Bei mehreren installierten OMD-Versionen die Standard-Version festlegen:

```
omd setversion
```

OMD konfigurieren:

```
omd config xenosite
```

Mit WATO einen Host hinzufügen, wo der Agent mit Hilfe von SSH abgefragt wird.

Configuration of Hosts and Services (Ruleeditor) -> Access to Agents -> Individual program call instead of agent access -> Create rule in folder OpenBSD-Server -> Edit rule Individual program call instead of agent access. In dem Feld Command line to execute habe ich folgendes eingetragen:

```
ssh -l christian <IP> check_mk_agent.openbsd
```

Python 2.7 kompilieren für Mac OS X 10.3.9 Panther damit Check_MK unter Panther funktioniert

Das kuriose ist, dass man OS X 10.4 Tiger benötigt, um für 10.3 Panther Python zu kompilieren. Also schnell Tiger gestartet. Dann die Python-10.3-Distribution erstellt und in die Icinga-Distri integriert. Und Check_MK funktioniert.

Check_mk: setup.sh on Mac OS X

The setup.sh script doesn't work on Mac OS X 10.6.8 Snow Leopard.

```
Error message:

sed: illegal option -- r
usage: sed script [-Ealn] [-i extension] [file ...]
       sed [-Ealn] [-i extension] [-e script] ... [-f script_file] ... [file ...]
```

There isn't a sed option --r on OS X. You need the option -E ;)

MRTG (Icinga-Distribution 1.7.X-X für Mac OS X)

Das Problem war, dass der Befehl nicht per Cron aufgerufen werden soll, weil das App keine Veränderungen im OS X-System machen soll. Ich habe einen kleinen Check gebastelt, der von Icinga aufgerufen wird und dieser dann die Graphen aktualisiert. Zusätzlich sehe ich noch in der Überwachung, ob MRTG läuft.

check_mrtg:

```
#!/bin/bash

env LANG=C /tmp/icinga.app/Contents/Resources/bin/mrtg
/tmp/icinga.app/Contents/Resources/etc/mrtg.cfg

if [ $? -ne 0 ]
then
echo "MRTG: Not Running"
exit 2
else
echo "MRTG: Running"
exit 0
fi
```

ssh-copy-id

Manchmal erscheint die Fehlermeldung:

```
/usr/bin/ssh-copy-id: ERROR: No identities found
```

Dann hilft der Aufruf mit der Option **-i** und dem öffentlichen Schlüssel:

```
ssh-copy-id -i id_rsa.pub christian@192.168.1.144
```

Wichtig: Unter OMD muss der öffentliche Schlüssel vom Site-User verwendet werden.

Mit WATO einen Host hinzufügen, wo der Agent mit Hilfe von SNMP abgefragt wird.

1. „WATO – Main Menu“ aufrufen
2. Schaltfläche „Hosts & Folders“ wählen
3. Schaltfläche „New host“ anklicken
4. Hostname eingeben
5. Option „IP address“ aktivieren und IP-Adresse eingeben
6. Option „Agent type“ aktivieren und im Pulldown-Menü „Legacy SNMP device (using V1)“ auswählen
7. Auf die Schaltfläche „Save & go to Services“ klicken
8. Services auswählen, die überwacht werden sollen
9. Schaltfläche „Save manual check configuration“ wählen