Research Report
Security Management with Open Source Tools

Prof. Dr. Stefan Wohlfeil

2006-12-20

This document describes the work done during the Research Semester in Summer 2006 of Prof. Dr. Stefan Wohlfeil. It is about *Security Management* tasks and how these tasks might be supported by Open Source software tools. I begin with a short discussion of general management tasks and describe some additional, security related management tasks. These security related tasks should then be added to a software tool which already provides the general tasks. *Nagios* is such a tool. It is extended to also perform some of the security related management tasks, too. I describe the new checking scripts and how Nagios needs to be configured to use these scripts.

The work has been done in cooperation with colleagues from the *Polytechnic of Namibia* in Windhoek, Namibia. This opportunity was used to also establish a partnership between the Department of Computer Science at FH Hannover and the Department of Information Technology at the Polytechnic. A first Memorandum of Agreement lays the groundwork for future staff or student exchange.

1 Introduction

1.1 Motivation

It is obvious that computers play an extremely important role in todays life. Not only private and personal use but also commercial use of computers require high security. Ecommerce companies rely on the fact that their computers are up and running 24 hours a day and 365 days a year. Security problems like viruses, worms, denial of service attacks prevent systems from performing their intended functions, thus creating massive economical losses. Therefore administrators need to make sure, that their systems are appropriately protected.

In general a secure system needs to provide these properties:

Confidentiality: This property requires that private information must remain private. Data containing this kind of information (trade secrets, economic data, strategic plans, etc.) should only be readable by authorized people (or authorized systems).

People not authorized to read the information should be unable to do so, no matter if the information is just stored in an IT system or electronically transferred, e.g. by email.

To ensure this property one can use *Access Controls* or furthermore *Encryption*.

Integrity: Information should remain unchanged or at least any change should be detectable by the owner of the information. This property is independent of the information being just stored somewhere or being in transit from one computer to another.

An attacker who changes an email which contains important information (e.g. an offer) should not go unnoticed.

Message Authentication Codes can be used to make sure that the integrity of data or messages is guaranteed.

Authenticity: This property means, that nobody else than some user itself might claim to be this user. Normally users of IT systems need an *account* to use the system. As anybody can try to use this account, additional measures are needed so that only the legitimate user can use the account.

Passwords are a technical means to prove identity. If *Public Key Cryptography* is available one can also use *Digital Signatures* to ensure authenticity.

Availability: IT systems should be available to their users whenever users need them. An ecommerce system like Amazon should be availabe around the clock so that customers can place their orders at any time.

Building a high-availability cluster is one of the technical measures to achieve this goal.

1.2 Security Technologies

In order to achieve Confidentiality, Integrity, Authenticity and Availability different techiques have been developed. Some of these are shortly introduced in this subsection.

Encryption: By encrypting messages users want to make sure that only authorized users can read the message [Sch96]. Encryption transform a message called *plain text* using a function controlled by a parameter called *key* into a ususally unreadable message called *cipher text*. The inverse function is called decryption function. Different encryption functions exist. They are also called *encryption algorithms*. *Symmetrical encryption algorithms* use only one key for encrypting and decrypting messages. Users need to agree on this key before they can confidentially exchange messages.

With *asymmetric encrytion* two different, yet related keys are used. One key is used to encrypt a message. This key is called *public key* because anybody needs to know this key to encrypt a message for somebody else. The receiver of the message possesses the second key which is called *private key*. This key is used to decrypt the message. Only the receiver knows the private key because nobody else should be able to read the message. It is important that messages encrypted with the public key cannot be decrypted using the public key. *RSA* is a well-known example of such a public key system.

Firewalls: The main purpose of a firewall is separating different networks and controlling all data passing from one network to the other [CZ95, Gon99]. Usually one of the separated networks is a trusted, internal network while the other network is an untrusted, external network (usually the Internet). Firewall rules describe which kind of communication between machines from the different networks is allowed and which communication should be blocked. Firewalls can operate on different ISO/OSI network layers. If they operate on the transport layer these firewalls are called *packet filtering router*. By using the information in TCP/IP packet headers (like source address, destination address, destination port number, etc.) the packet filter decides if a packet is allowed to pass. Another kind of firewall, called *application level gateway* or *proxy* operates on the application layer of the protocol stack. They can use additional information not present in TCP/IP headers like which user is logged in or which command was sent, etc.

Therefore firewalls should be able to block some attacks originating from the Internet and directed to the internal network. As firewalls are not perfect and attacks might also come from internal machines, additional security measures are required.

Intrusion Detection: The purpose of an Intrusion Detection System (IDS) is to recognize attacks and inform system administrators about the attack [NN01, Spe03]. The administrator can then take appropriate action.

Intrusion Detection Systems can work in two different ways:

1. *Recognize already know attacks:* In this mode, the IDS can alert its user as soon as it sees a sequence of events which is known as an attack pattern. This mode produces very few false positives, i.e. alarms which do not belong to an attack. The drawbacks of this mode are: (1) It can only detect attacks already known. New and unknown attacks cannot be detected. (2) The user of the IDS needs to regularly update the attack patterns, so that all know attacks can be recognized.

2. *Anomaly Detection:* Here the IDS checks if the events that occur belong to normal use of the system. Everything which is not normal is suspicious and might be an attack. A big problem here is, finding out what is normal. This mode produces more false positives because all changes in system use raise an alarm, even if the change is legitimate.

The events that an IDS uses to recognise attacks might also be twofold:

1. Network events like IP packets travelling around. This kind of IDS is called *Network Based IDS*. It looks into all packets travelling in a network and looks for malicious content. Using this method, attacks can be found even before the packets have reached their target. However attacks which are performed without using the network, e.g. attacks by somebody sitting at the console, cannot be detected.

   Snort is an open source network based intrusion detection system that uses patterns to detect attacks.

2. Host events like changed files or system calls. This kind of IDS is called *Host based IDS*. Integrity checking tools create a snapshot of the important system files just after the system has been installed. This snapshot is a list of file names and *Message Authentication Codes*. The IDS regularly compares the current version of the important system files with the snapshot. If a file has changed, then an alarm is raised. The programs *tripwire* and *aide* do just that.

   Other Host based IDS change the kernel of the operating system and intercept all system calls. By analysing the system calls, these system can see what happens on the computer. If some dangerous system calls are made, then an alarm is produced.

2 Management Functions

Most networks have an *administrator* who is responsible for the network, i.e. that the network and all important computers in the network are up and running. If problems occur, then the administrator will work on solving them. Problems might occur for different reasons:

- Hardware failures, e.g. because of too much heat in the summer.
- Software failures, e.g. because the operating system or a program crashes.
- Denial of Service Attacks, e.g. when a hacker actively attacks a system to produce software failures or something similar.

The first to reasons have occured since computing has started. With SNMP and RMON [Sta00] two protocols have been developed to support network management functions. Administrators are used to these kinds of problems and monitoring tools exist which address these issues. Denial of Service Attacks have only rarely occured before the Internet became popular. Today, anybody with Internet access can easily start attacks on other systems connected to the Internet. And as most of the (important and interesting) computers are connected to the Internet, they might be targets for this kind of attack. Intrusion Detection Systems (IDS) have been built to deal with attacks. However they are not integrated into the monitoring tools which already exist.

2.1 General Management Functions

System Availability: The first general management operation is about finding out if a network or a system is still available. To do this one can use the *Internet Control*

Message Protocol (ICMP). For example, the ping command sends an ICMP packet to a machine and waits for the answer. If the command was successful, then one can deduce that the network is working and the system is also up and running. To be more precise, one knows that the *Operating System (OS)* is up and that the networking part, usually the so called TCP/IP stack, is running.

Service Availability: The next step is to check if the system is also capable of doing what it should do, i.e. providing the services it should provide. A web server, for example needs to answer *HyperText Tranfser Protocol (HTTP)* requests. Besides the system itself being available this also requires that the process which should provide the service is up and running. A web server process, e.g. an apache process, might provide the HTTP service.

To check for service availability one can try to use the service regularly. A check process running on one machine can try to get an HTML page from a web server. If the server sends the page, then the service is still running. Otherwise the check should send a problem report.

2.2 Security Management Functions

Besides the traditional management functions there are some management functions related to security.

Installed Security Patches: On a secure system all security patches provided by the manufacturer of the operating system should be installed. Therefore administrators need to make sure that these updates will be installed. Most update programs provide mechanisms to automatically download, check and install updates. Very often administrators like to test updates before installing them, so automatic update functions are often deactivated. Therefore administrators might forget installing an update.

Integrity Checking: A system which has been hacked successfully is usually manipulated by the hacker. This manipulation should allow later use of the system. Hackers install so called *back doors*. These might change important configuration files or manipulted important system programs, e.g. the login program. One can recognize such manipulations by inspecting the files of that system.

System integrity also requires that all security updates which are available for this system are installed. Administrators can either install these updates manually or configure a system to automatically install the updates. In either case it is necessary to check that all updates were really installed.

Installed Services: If a hacker wants to connect to a computer, usually network connections are used. Therefore the hacker needs to use a network service on the computer. Some back door programs provide such services by starting a service process which waits for incomming connections to a particular *port number*. One can use port scanners to check for such additional services processes.

3 Implementing Management Functions

3.1 Open Source Management Tools

Yast Online Update (YOU): This program is provided by Novell/Suse to perform online updates of the operating system. The program can also be used to check, if all security updates and all recommended updates were installed. We will use this feature in a Nagios script to check the updates on our servers. Jeff Falgout wrote a Perl script `check_suse_updates.pl`. As we use SuSE Linux Enterprise Edition 9 we need to be registered customers in order to download updates. We use Jeffs script therefore as follows:

```
check_suse_updates.pl -u https://username:password@you.novell.com/update
```

Other OpenSuSE versions do not need usernames and password. As they might provide their updates on different servers, the URL in the above command needs to be adjusted.

tripwire/aide: These programs are file system integrity checkers. They compare the current state of a file system with a previously saved state which is known to be consistent. As soon as some files have changed, e.g. the password file, then this change might indicate a successful attack.

nmap: The program *nmap* is a port scanner. It tries to open TCP or UDP connections to a computer. A securely installed computer should only accept connections to some services. If an nmap port scan shows additionally activated services on a computer, e.g. some kind of backdoor service, then this indicates a security problem. Checking computers on a regular basis using nmap is therefore a security management function. Checks are performed as follows:

1. The administrator creates a first nmap scan of the target host by issuing the command `checkwithnmap -c ComputerName ScanResult`
2. Now the file `ScanResult` can be used in the following regular scans of the computer `ComputerName`. Nagios is configured to start the command `check-with-nmap ComputerName ScanResult`

As nmap scans might take a while (around one minute or so) the schedule of this check needs to be adjusted. At most every hour performing this check should be sufficient. The `checkwithnmap` script looks like this:

```
#!/bin/sh
# File: check-with-nmap Skript
# Author: Stefan Wohlfeil
# Date: 2006-11-22
# Purpose: Check a remote computer automatically with nmap
# History: 2006-11-22 Version 1.0
```

```
#
# Return codes and their meaning:
#  0 OK
#  1 Warning but not critical
#  2 Critical or plugin has timed out
#  3 Unknown: some error within the plugin

if [ $# -eq 2 ]
then
   rechner=$1;
   nmapfile=$2;

   if [ ! -r $nmapfile ]
   then
      echo "File $nmapfile does not exist or is not readable"
      exit 2;
   fi
# grep filters out the Greeting line of nmap and the last line, too.
# These lines are always different because they contain the date and time
# when nmap was started and how long the scan took.
   nmap $rechner | grep -i -v Nmap | diff $nmapfile - > /dev/null
   result=$?;
   if [ "$result" == "0" ]
   then
      echo "Open Ports OK - Nothing has changed"
      exit $result
   else
      echo "Open Ports ERROR - Something has changed. Check with comand $0"
      exit 2
   fi
elif [ $# -eq 3 ]
then
   if [ "$1" == "-c" ]
   then
      rechner=$2;
      nmapfile=$3;

      if [ ! -w $nmapfile ]
      then
         echo "File $nmapfile cannot be written to."
         exit 2;
      fi
      nmap $rechner | grep -i -v Nmap > $nmapfile
   else
      echo "Usage: ‘basename $0‘ [-c] MACHINE-NAME EXPECTED-OUTPUT-FILE";
      exit 3;
   fi
else
   echo "Usage: ‘basename $0‘ [-c] MACHINE-NAME EXPECTED-OUTPUT-FILE";
```

```
   exit 3;
fi
```

Nagios: On its web site the developers of Nagios describe their tool as follows:

> Nagios is a host and service monitor designed to inform you of network problems before your clients, end-users or managers do. It has been designed to run under the Linux operating system, but works fine under most *NIX variants as well. The monitoring daemon runs intermittent checks on hosts and services you specify using external "plugins" which return status information to Nagios. When problems are encountered, the daemon can send notifications out to administrative contacts in a variety of different ways (email, instant message, SMS, etc.). Current status information, historical logs, and reports can all be accessed via a web browser.

So Nagios is the management tool that I will use as the basis for the security management. By configuring it appropriately and extending it, some of the security management functions were implemented. Figure 1 shows a screen shot of the Nagios installation at the Department of Computer Science at FH Hannover.

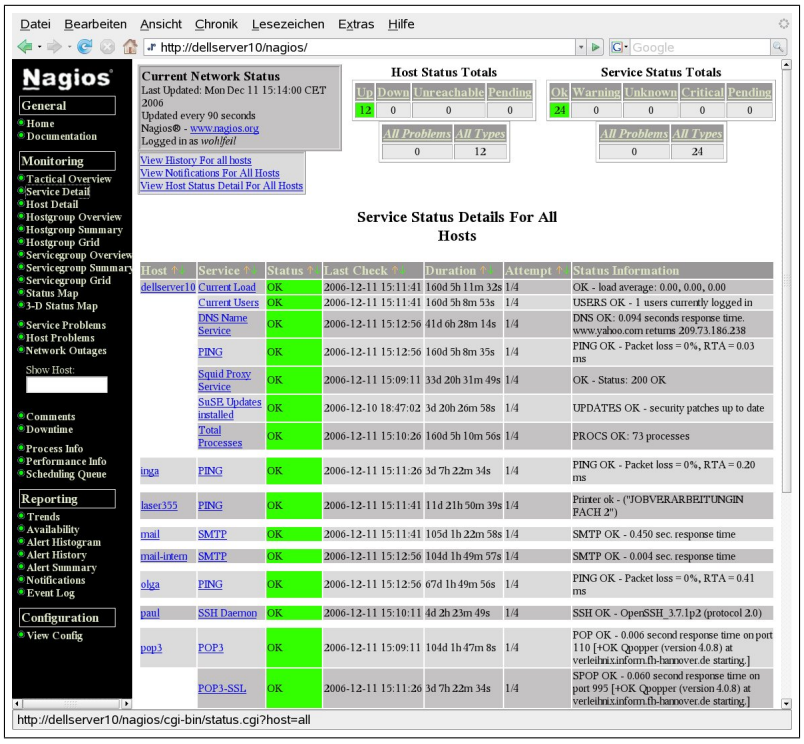

Figure 1: Nagios Screen Shot

3.2 Installing, Configuring and Extending Nagios

In this subsection all steps to set-up Nagios are explained. It begins with how to download and install Nagios, continues with how it can be configured and finishes with an explanation of the extensions which were made for the security management tasks.

Installing Nagios: In order to get an appropriately configured Nagios [Bar05] implementation, some special care needs to be taken. In this section the installation process is described in detail. We begin with the general installation procedure and after that we describe the configuration.

1. Download the current version of Nagios from `http://www.nagios.org/`

2. Unpack the software:

   ```
   cd Software
   tar xzvf ../Download/nagios-2.4.tar.gz
   ```

3. Create a Nagios user and a Nagios group. Add the user to the group.

   ```
   useradd nagios;
   groupadd nagios;
   usermod –G nagios nagios;
   ```

4. Create a group for Nagios commands.

   ```
   groupadd nagcmd;
   ```

5. Add the user id of the web server process (wwwrun) to the group Nagios commands. This is necessary, so that users can start commands using the web interface. In that case, the web server process starts the commands on behalf of the user.

   ```
   usermod –G nagcmd wwwrun;
   usermod –G nagcmd nagios;
   ```

6. Configure Nagios, so that it is installed in /opt/ and so that the users and groups created above are used. Compile the program and install it.

   ```
   ./configure --prefix=/opt/nagios --with-cgiurl=/nagios/cgi-bin \\
      --with-htmurl=/nagios --with-nagios-user=nagios \\
      --with-nagios-group=nagios --with-command-group=nagcmd
   make all
   make install
   ```

7. Tell the web server that the Nagios web interface is available. Create a file `nagios.conf` in`/etc/apache2/`.

```
ScriptAlias /nagios/cgi-bin /opt/nagios/sbin

<Directory "/opt/nagios/sbin">
    Options ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /opt/nagios/etc/htpasswd.users
    Require valid-user
</Directory>

Alias /nagios /opt/nagios/share

<Directory "/opt/nagios/share">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthName "Nagios Access"
    AuthType Basic
    AuthUserFile /opt/nagios/etc/htpasswd.users
    Require valid-user
</Directory>
```

Now tell apache that this file is available by adding

```
Include /etc/apache2/nagios.conf
```

to the file `httpd.conf`.

As the web server is also responsible to authenticate users, these users need to be added.

```
htpasswd2 -c /opt/nagios/etc/htpasswd.users nagiosadmin
htpasswd2 /opt/nagios/etc/htpasswd.users wohlfeil
```

8. As the software package only contains the basic Nagios software, some additional Nagios plug-ins need to be installed. This is done as follwos:

```
tar xzvf ../Download/nagios-plugins-1.4.3.tar.gz;
cd nagios-plugins-1.4.3;
./configure --prefix=/opt/nagios
make all
make check
```

When doing the check, the appropriate computer names or IP addresses need to be provided.

9. Create a new configuration file `fbi.cfg` in Nagios `etc` directory. Change the standard configuration `nagios.cfg` such that `fbi.cfg` is read by `nagios.cfg`. The file `fbi.cfg` contains all the special information about the Fachbereich Informatik (FBI). Within `nagios.cfg` also change the date format to ISO standard using a line like `date_format=iso8601`

Configuring Nagios: The Nagios server of the Department of Computer Science at FH Hannover monitors the departments server machines. The Nagios server itself is located in the Intranet of the Department. This Intranet is separated from the University network by a firewall. The firewall uses the classical Demilitarized Zone (DMZ) architecture. Therefore some of the monitored machines are located behind packet filters (routers). See figure 2 for details.

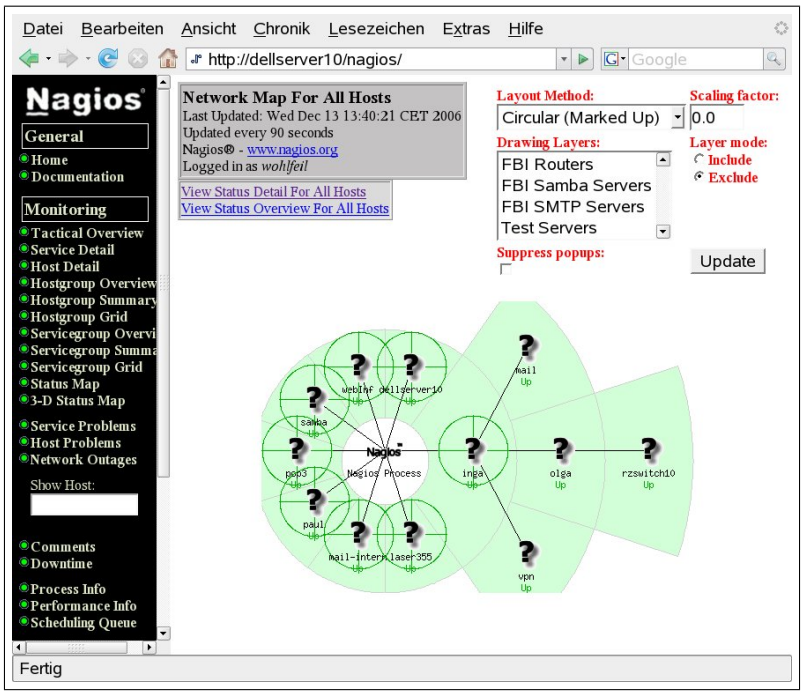

Figure 2: The Nagios Map of the monitored machines

Nagios should be used to check if all security updates are installed on the Nagios server itself. In order to do so, these steps are required:

1. Write a check function `check_suseupdates_ds10` in file `checkcommands.cfg` as follows:

```
define command {
  command_name    check_suseupdates_ds10
  command_line    $USER1$/check_suse_updates.pl -u https://xxx:yyy@you.novell.com/update/
}
```

2. Define a service check in the file `fbi.cfg` as follows:

```
define service {
  use                          generic-service ; Name of service template to use
  host_name                    dellserver10
  service_description          SuSE Updates installed
  is_volatile                  0
  check_period                 24x7
  max_check_attempts           4
  normal_check_interval        1440
  retry_check_interval         60
  contact_groups               swohlfeil
  notification_options         w,u,c,r
  notification_interval        1440
  notification_period          24x7
  check_command                check_suseupdates_ds10
}
```

Now Nagios automatically checks, if all security updates are installed. Doing the same check on a machine different from the Nagios server itself requires more work. Before checking one needs to open a SSH connection to the machine which should be checked. Our VPN gateway is an example of such a machine. To perform the check one has to do:

1. Write a checking function is file `checkcommands.cfg` as follows:

```
define command{
  command_name    check_suseupdates_vpn
  command_line    $USER1$/check_by_ssh -H $HOSTADDRESS$ -i /opt/nagios/etc/.ssh/id_dsa
       -t 60 -C "/usr/lib/nagios/plugins/check_suse_updates.pl
       -u http://dellserver10.inform.fh-hannover.de/suseupdate"
}
```

   This check function uses the `check_by_ssh` function. This function connects the Nagios server to a remote machine and then performs the command line given as parameter -C on the remote machine. In order to open the connection without entering a password we use public key authentication.

2. Define a service for this check in file `fbi.cfg` as follows:

```
define service{
```

```
    use                         generic-service  ; Name of service template to use
    host_name                   vpn
    service_description         SuSE Updates installed
    is_volatile                 0
    check_period                24x7
    max_check_attempts          4
    normal_check_interval       1440
    retry_check_interval        60
    contact_groups              admins
    notification_options        w,u,c,r
    notification_interval       1440
    notification_period         24x7
    check_command               check_suseupdates_vpn
}
```

Extending Nagios: With nmap we can quickly check, if additional services have been installed on a server. To do this, we save the output of nmap when the server has been checked. This output contains information about which ports are open. A Nagios script then calls nmap regularly and compares the output of each run with the saved output. However the nmap output contains a first line with the date and time when nmap was started. This line always differs between two runs of nmap. The same is true for the last line. nmap writes the time it took to perfom the scan in the last line. Before comparing the files, these two lines need to be filtered out. This is automatically done by the script presented in section 3.1.

The script is installed in the Nagios directory where all other scripts are installed, too. Then the administrator needs to create a default output file of the nmap check of one machine by issuing these commands:

```
touch computer-name-output.txt
checkwithnmap -c computer-name computer-name-output.txt
```

As the script requires that the output file already exists, the touch command creates an empty output file. Then the check script creates the filtered output. The administrator should check the output to find open ports which should not be open. Then the administrator can add the regular check to Nagios. First a Nagios command is created:

```
# Command to check open ports on a machine
# ARG1 = Name of file with the "normal" nmap output
define command{
   command_name check_with_nmap
   command_line $USER1$/checkwithnmap $HOSTADDRESS$ $USER1$/$ARG1$
}
```

The check script and the normal output file should be located in the same directory. The variable USER1 hold the directory name. The second argument of the Nagios command then specifies the file name. The regular service check is then configured:

```
define service{
```

```
    use                     generic-service ; Name of service template to use
    host_name               paul
    service_description     Open Ports
    is_volatile             0
    check_period            24x7
    max_check_attempts      4
    normal_check_interval   60
    retry_check_interval    10
    contact_groups          admins
    notification_options    w,u,c,r
    notification_interval   960
    notification_period     24x7
    check_command           check_with_nmap!paul-normal.txt
}
```

In our case the computer named `paul` is checked an the file with the expected nmap output is called `paul-normal.txt`.

4 Conclusion

4.1 General results

This work has shown that adding security extensions to Nagios and configuring Nagios appropriately allows performing some security management functions. By adding the nmap script, a Nagios server can (1) automatically check, if no additional services have been added to a server and (2) that all services which are expected to be available on the server really are available. Hackers installing a backdoor wil be recognized fast and appropriate action can follow.

Nagios can also automatically check, if all security patches of a SuSE Linux system are installed. If this script fails or notices that updates are missing, an alert is sent to the administrator. This is useful if administrators don't want to automatically install security updates. Reasons might be, that automatical updates might require a reboot and the administrator doesn't want unattended reboots. On mission critical machines administrators often also prefer checking security updates before installing them. Now Nagios offers an alert mechanism which tells administrators when they need to work on security updates on their machines.

4.2 Frist practical experiences

We use Nagios with the described extensions at the Department of Computer Science of Fachhochschule Hannover. Especially the script which checks for available updates has proven to be very useful. As security updates were published in irregular time intervals it is convenient to have such an automatic check. Although all operating system vendors provide mailing lists where security updates were announced, the script is still very useful. The reason is that *all* updates are announced on these lists, even if the update is for a package not installed in our configuration. The script only checks for updates of packages installed.

A second advantage of the script is, that the announcement often is sent a few days after the patch has been distributed to the download sites. So the scripts might notify about these updates earlier than the distributor does on its mailing lists.

Experiences with the nmap script are also positive. It is used to closely monitor our SSH server. This server can be contacted from the Internet and is therefore open for attacks. The regular port scans would quickly show if a hacker had successfully hacked the machine and installed some backdoor processes.

The only drawback of this script is, that it generates false alarms in our Intrusion Detection System. Our IDS can recognize port scans as an attack preparation. However the nmap port scan is a discovery action so it should not generate an alert with the IDS. So we need to either ignore these alarms or reconfigure our IDS.

4.3 Future work

The current extensions of Nagios are only a first step towards automatic security management. We plan to add more management functions to Nagios. There are at least two more useful management function

1. Check the integrity of a remote computers hard disk. This requires creating a Nagios plug-in which can use tripwire oder aide to recognize potentially dangerous changes to the file system.

2. Check the log files of a remote computer for suspicious entries. This requires a plug-in which uses software like logsurfer.

Moreover we also plan to manage Microsoft Windows Computers by integrating the *Microsoft Baseline Security Analysis (MBSA)* tool into Nagios.

5 Cooperation with Polytechnic of Namibia

The Department of Computer Science has agreed to sign a memorandum of understanding with the Polytechnic of Namibia. It is the first step towards a closer working relationship. Its goals are exchanging staff or students between the institutions.

5.1 The Memorandum of Understanding

MEMORANDUM OF AGREEMENT
BETWEEN
POLYTECHNIC OF NAMIBIA
13 Storch Street
Private Bag 13388
Windhoek
NAMIBIA
Telephone: (+264-61) 207- 2000/1/2/3
Facsimile: (+264-61) 207-2100

AND

Fachhochschule Hannover
Department of Computer Science
Ricklinger Stadtweg 120
30459 Hannover
GERMANY
Telephone: (+49-511) 9296-1800/1/2
Telefax: (+49-511) 9296-1810

This MEMORANDUM OF AGREEMENT outlines a program of cooperation between the Department of Information Technology at the Polytechnic of Namibia hereinafter referred to as the POLYTECHNIC and the Department of Computer Science at the Fachhochschule Hannover hereinafter referred to as the FHH/CS.

WHEREAS the POLYTECHNIC and the FHH/CS recognize the value of educational cooperation and exchanges for the purpose of deepening the understanding of scientific, technological, historical, social, economic and political issues, as well as the traditions of each of the respective cultures; and WHEREAS the POLYTECHNIC and the FHH/CS affirm the desirability of strengthening the bonds between the two academic communities through the development of curricula, exchange and secondment of faculty, staff and students and research at both institutions.

IT IS MUTUALLY AGREED AS FOLLOWS:

In order to promote international co-operation and understanding and to foster institutional development, the two institutions agree to co-operate in the areas of: curriculum development, faculty, staff and students exchange; research and materials exchange.

1. CURRICULUM DEVELOPMENT

   POLYTECHNIC and the FHH/CS agree to develop and upgrade curricula and qualifications with the objective of offering quality programmes and qualifications subject to the approval by the respective Senates.

2. FACULTY/STAFF EXCHANGE

   In order to promote professional development at their respective institutions, the institutions agree to establish an exchange program for faculty and professional staff. The host institution agrees to provide hospitality in the form of office space, library privileges, access to technology and resources and other local privileges customarily afforded to faculty in residence.

   The host institution will also assist in the finding of *suitable accommodation* and assist with compliance with immigration regulations of the host country. Faculty exchanges will be mutually agreed upon between the institutions and will be based on a work plan covering the full period of the exchange. The visiting scholar will prepare a final report upon completion of the exchange period.

3. STUDENT EXCHANGE

   The two institutions will develop and conduct a student exchange program offering coursework, internships, joint projects, and research which are directed at the preparation of highly qualified and internationally competent graduates in Computer Science and Information Technology.

   Participants will be registered students, selected by their respective institutions to participate in the program on the terms agreed between the institutions.

   In general, English will serve as the language of instruction at both institutions during exchange periods. However, language programmes will be developed in German to help students cope with language demands while on exchange.

4. RESEARCH AND MATERIALS EXCHANGE

   The POLYTECHNIC and FHH/CS will encourage and promote collaborative research activities between the faculties, staff and students of the two institutions to promote the generation of knowledge and development of the institutions and their respective societies. This will include upgrading of laboratory facilities and the installation of specialized equipment for relevant departmental laboratories.

5. FORMAL FACULTY AND STUDENT DEVELOPMENT OPPORTUNITIES

   To further build human capacity in Department of Information Technology at the Polytechnic of Namibia the FHH/CS will offer nominated staff and students opportunities to obtain formal qualifications in the full-time mode at Bachelors and Masters degree levels. The specific terms and conditions will be agreed upon in an addendum that will form part of this agreement.

6. FUNDING

   The POLYTECHNIC and FHH/CS agree to investigate opportunities to raise funds to support the different activities in the partnership. POLYTECHNIC and FHH/CS agree to facilitate activities with Namibian business and industry when appropriate, and vice versa.

7. PROGRAM COORDINATORS

   As specific and mutual areas of interest are identified and agreed, each institution will appoint a program coordinator to assure the implementation of the various planned activities.

8. IMPLEMENTING AGREEMENTS

   Supplemental agreements will be entered into to establish details for the operation and implementation of specific programs and will become attachments to this Memorandum of Agreement.

9. DURATION OF AGREEMENT

   This Memorandum of Agreement will be effective upon signature by duly authorised representatives and will be effective for a period of five years.

10. AMENDMENT OF AGREEMENT

    This Memorandum of Agreement may be amended by mutual consent, in writing, as appropriate from time to time.

11. TERMINATION

    This Memorandum of Agreement may be terminated by either party following a 90-day written notice. However, the effective date of cancellation shall not take place prior to the completion of an on-going activity.

Signed at Windhoek, NAMIBIA, on this..... day of ..... 2006.

FOR: FH Hannover, Department of Computer Science	FOR: Polytechnic of Namibia
..... Dean	..... Rector
WITNESS:	WITNESS:
.....	.....

References

[Bar05] Wolfgang Barth. *Nagios — System- und Netzwerk-Monitoring.* Open Source Press, 2005.

[CZ95] D. Brent Chapman and Elizabeth D. Zwicky. *Building Internet Firewalls.* O'Reilly & Associates Inc., Sebastopol, CA, 1995.

[Gon99] Marcus Goncalves. *Firewalls: A Complete Guide.* McGraw-Hill, October 1999.

[NN01] Stephen Northcutt and Judy Novack. *IDS: Intrusion Detection. Spurensicherung im Netz.* mitp, 2001.

[Sch96] Bruce Schneier. *Angewandte Kryptographie. Protokolle, Algorithmen und Sourcecode in C.* Addison-Wesley, Bonn, Germany, 1996.

[Spe03] Ralf Spenneberg. *Intrusion Detection für Linux-Server.* Markt+Technik Verlag, D-81829 München, Deutschland, 2003.

[Sta00] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Addision Wesley Longman, Reading, Massachusetts, USA, 2nd edition, 2000.

[Woh06] Stefan Wohlfeil. *Kurs 1867: Sicherheit im Internet 2.* FernUniversität Hagen, Hagen, Germany, 2nd edition, 2006.