THE IMPLEMENTATION OF THE VITELS IP SECURITY DISTANCE LEARNING MODULE

Computer Science Project

Reto Gantenbein
April 2007

Head:
Prof. Dr. Torsten Braun

Assisted by:
Thomas Bernoulli, Thomas Staub

RVS (Computer Networks and Distributed Systems)
Institute of Computer Science and Applied Mathematics (IAM)
University of Bern

Contents

Introduction

The VITELS distance learning module 'IP Security' is hosted by the RVS group at the University of Bern. It already went and will go again through different minor and major changes. The main goal of this project in computer science was to migrate the bootstrapping process of the Cisco routers used in this module to a bootstrapping process using a TFTP server. Because it is relatively simple to implement this change, it was further asked to exchange the hardware back-end of the 'IP Security' module by new machines and to document the whole installation. A sample solution for the practical exercise session of this module should also be provided. Therefore this document should be used to troubleshoot the installation and give support to the module maintainer. The setup documentation will help to integrate the module into a new Java-based Web portal software which is currently under development and is going to replace the current PHP-based portal.

Document Structure

At first there is a short introduction into VITELS and the an overview of the module (chapter 1). It is followed by the hardware description (chapter 2), an installation guide for the operating system (chapter 3) and the scripts on all the involved machines (chapter 4). Afterwards the TFTP setup is explained (chapter 5). A solution for the practical module part, the so called 'Hands-On Session' follows (chapter 6). Thanks to my employment as teaching assistant on the Computer Networks lecture, where this module was extensively used by the students, the experience with this module was written down to inspire future improvements (chapter 7). All used custom scripts are attached in the appendix C.

Text Formatting

In this documentation different formatting styles are used to accent different terms:

- Words that are printed *cursive* mean files or paths. e.g. */home/host1/*, *module_reset.sh*
- Words that are printed in `typewriter` font mean commands on the Linux or router console. e.g. `ls -la`.

  There can also be whole sections in typewriter font which are step-by-step instructions for the command line:

```
mkdir /home/host1
cd /home/host1/
```

- Listings in a grey box are printouts of configuration files. Depending on the caption it can be a whole file or only a section of it:

```
search unibe.ch
nameserver 130.92.64.4
```

Listing 1: /etc/resolv.conf

When using program and institution names or brands they are written in their common notation (e.g. only capital letters).

Chapter 1

Vitels

VITELS stands for 'Virtual Internet and Telecommunication Laboratory of Switzerland' and was developed in the last few years on several Swiss academies as instrument for Web-based learning. It provides a modular structured on-line course covering a variety of topics in the area of telecommunications and computer networks. Besides an established theory part, it also focuses practical experience in the considered domains. VITELS consists of eight modules: 'Simulation of IP Network Configuration', 'Client/Server Concepts', 'IP Security', 'Firewall Management', 'Sockets and Remote Procedure Calls', 'Remote Method Invocation', 'Application Server' and 'Linux Installation' that are provided by the universities of Geneva, Fribourg, Neuchâtel, Bern and the University of Applied Science of Fribourg. At the University of Bern the VITELS modules are under constant use of the Computer Networks Laboratory and the Computer Networks lecture.

VITELS was one of the first e-learning applications which was connected to the Swiss-wide authentication and authorisation infrastructure AAI, lead by SWITCH the Swiss Education and Research Network. SWITCH arose from a collaboration of numerous Swiss universities and provides today, among other things, a high speed network between the universities in Switzerland and connections to the Internet and other global science networks.

The basic principle of this authentication system is as follows: A client (e.g. a student) is interested in an e-learning content that is provided by one of the VITELS partners. When he wants to access it, the authentication infrastructure redirects him to the authentication server of his own university, and after successful authentication, it grants permission to the desired resource.

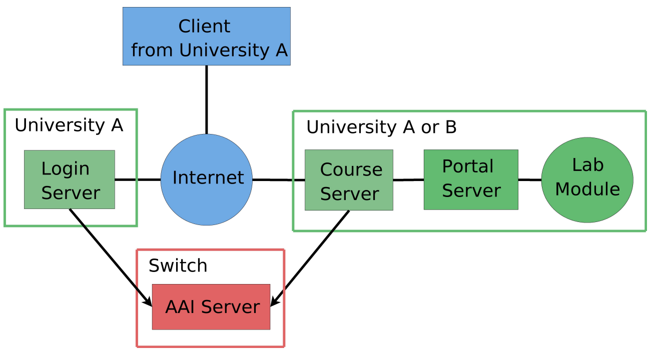

Figure 1.1: AAI/VITELS Architecture

The VITELS modules are educational structured into four parts. The first part is an introduction into the topic and module. It is followed by a theory part and a knowledge application/exploration part. The latter contains the practical hands-on session. The modules are then completed by a final quiz part. Every module is structured the same way what helps the effective learning and testing of the gained knowledge. During the entire preparation of the module, the student can make notes into a logbook and has to fill small in-between tests and essays. Like that, he has all the time an overview about what he already learnt and the tutor immediately receives feedback from the student.

1.1 Vitels IP Security Module

The theory part in the 'IP Security' module introduces the reader into Virtual Private Networking and an implementation of it called IPSec (RFC4301). For more information about the reading section check the Computer Science Project 'IP Security Module for VITELS' of Thomas Spreng [1]. The reading part is finished with a first theory quiz. After that the student has to solve the knowledge application part which will be especially discussed in this chapter. For the hands-on session we have a small independent network with two routers, three hubs and three Linux hosts. The student can access a console of each device over a Web portal and has to setup and test some given configurations. He should then store its practical progress by copying the console content to the logbook. In the final quiz part, a test about the practical session has to be passed, a survey filled and a final report about the personal synthesis has to be given.

The target for the student is to gain some experience with Cisco routers and gain some knowledge about and experience with network protocol security. The exact module content and the solutions of the exercises can be found in chapter 6.

Chapter 2

Hardware Installation

2.1 Requirements

The hardware requirements for the 'IP Security' module are rather moderate. There are no special processor, memory or storage size or performance requirements. The most calculation intensive part is the VPN decryption and encryption that is done by the processors of the routers. For the hosts and the portal server we took four Intel Celeron 1.8 GHz machines with 512 MB of RAM and a 80 GB hard disk. If the computers are in a rack-desktop case, they can easily be installed into a 19“-rack. Like that, two complete modules fit into a normal 42 U rack. To save some space and hardware resources it would even be possible to run all the hosts on a single machine, for example with the virtualisation software Xen [5] or User-Mode Linux [6].

The most important features our portal server has to provide are the I/O-ports. The setup needs three serial connectors; two for the terminal connections to the routers and one for a relay card to switch the power supply of the routers. Therefore the standard serial port on the computer mainboard has to be extended by a PCI serial interface adapter with two ports. Furthermore four network ports are required for connecting three hubs and the university network to the portal server. Here, a standard PCI four-port FastEthernet adapter fits our needs.

For the network we need three multi port repeaters (hubs) with at least four ports and two Cisco routers. We have two nearly identical module setups one with two Cisco 2600 routers and the other with two Cisco 3600 routers. For our scenarios the configuration of both of these router types is identical.

A special device is the relay card which controls the electricity supply of the routers. It should be able to switch on and off two separate 230V power lines over the serial connection. Therewith the routers can be hard reset by interrupting the power connection. The relay we used is available as construction set at Conrad Electronics [7].

Not really required, but useful, is a KVM (keyboard-video-mouse) switch. We installed a twelve-port switch to have direct system maintenance access to all the hosts and portal servers of the two modules.

2.2 Architecture

The network is divided into three subnets each using a hub as core distribution unit. Every host belongs to one subnet and is directly connected to the repeater. The three hubs are linked to each other by the two routers which direct the network traffic to the correct destination subnet. The portal server is separately connected with every hub and as the only computer with the university network and the Internet. There are serial connections for the configuration console between the portal server and the two routers. The third serial connection is used for switching the relays which control the power line of the routers' power supplies.

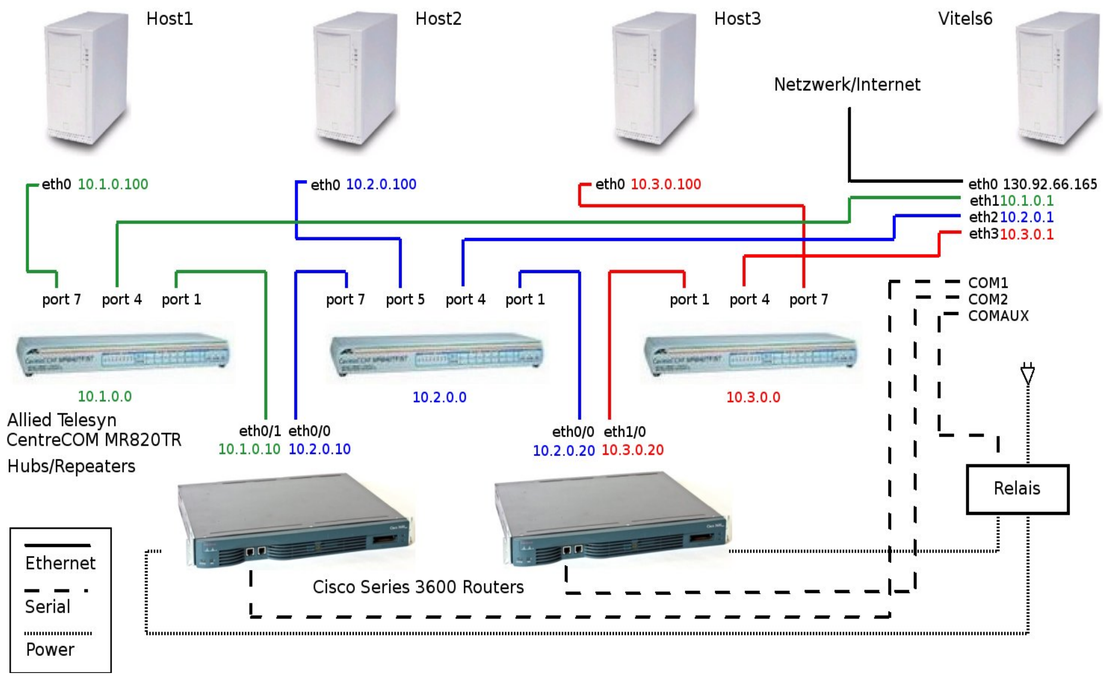

Figure 2.1: Network Details

Chapter 3

Linux Installation

This chapter describes the Linux operating system installation. All four machines of the module have the same basic setup. The easiest way to achieve this is to install only one machine and clone its installation to the others.

3.1 Basic Installation

The operating system plays an important rule of a stable operation of the 'IP Security' module because the computers would run for a longer period without extensive system administration. The original installation was running on the approved Debian GNU/Linux. Since the new stable version 3.1 (Sarge) was released in 2005, when the new hardware was set up, this was the best choice for our new machines too.

For the Sarge release the developers busily worked on implementing a comfortable installer. They succeeded and so it is an easy task to setup a Debian server. For the exact installation instructions follow the next lines. As already mentioned before, this installation is, except from the network configuration, the same for all the Linux machines in our network.

Fetch the newest CD image of the Debian release from a mirror given by debian.org (in our case debian-31r0a-i386-binary-1.iso). After we burnt it on CD, the computer can directly boot from it. Then a few questions have to be answered:

Language - English

Country - Switzerland

Key Map - As keyboard layout we choose 'Swiss (German)' (ch_DE). If we want to change it again later, we have to execute `dpkg-reconfigure console-data`.

Host Name - It is the name of the computer in the network. Choose from gridlab06 (module 1) or gridlab07 (module 2) for the portal servers and host1, host2 or host3 for the remaining hosts. The name is stored in */etc/hostname*.

Domain Name - At least the portal server is reachable from the Internet therefore we fill in 'unibe.ch' which is added as first entry to */etc/resolv.conf*.

Hard Disk Partitioning - Select 'Manually edit partition table'. Because we have no special data on our hosts, the partitioning is more or less simple. We do not need the entire 80 GB therefore we leave the unused space unpartitioned for possible future use. A separate */boot*-partition is made for the kernel image and the boot loader. For security reasons this partition is not mounted by default. The same applies to the */backup*-partition that could be used for storing some copies of the machine configuration. The */var* is also on a separate partition because its logging content can grow seriously and suddenly fill the partition. Of course we do not want to crash our machines because of this. For the exact */etc/fstab* file see Appendix B.1. Finally the partition table looks like this:

Device	Mount point	Size	Partition type
/dev/hda1	/boot	100MB	Linux
/dev/hda2	-	500MB	Linux swap
/dev/hda3	/	7GB	Linux
/dev/hda5	/var	4GB	Linux
/dev/hda6	/backup	10GB	Linux

Table 3.1: Harddisk partitioning

File System Formatting - For our purpose the file system type does not play an important role. To prevent problems with the boot loader the */boot*-partition is formatted with ext3 [8]. On the other partitions the more modern XFS [9] is used. It has a good performance, there are a lot of administration tools for XFS and it is widely supported today and therefore it is a good choice in every case.

Boot Loader Installation - The machines have no complex hard disk setup. Therefore the default boot loader GRUB [10] can be installed into the master boot record (MBR).

Timezone - CET - Central European Time

Root Password - Choose a good root password. At minimum 8 characters with digits, letters and special characters.

User Creation - Select a user name and a password. After the installation this user can be used as default user. Later we will disable direct root login. For security reasons, we then first have to log in with the just created user before we can gain root privileges.

Choose package archive source - On the subnet hosts (host1-3) we choose 'cdrom'. All available CDs have to be inserted to let the package manager know which programs are available to it. For our installation it is enough to hold CD1 ready. On the portal server a lot more packages were needed and thanks to Internet access we can choose 'ftp' or 'http' here.

3.2 Additional Software Installation

In the last part of the installation we are asked if we want to install some more packages. Normally you would choose here a preselection of packages for special purposes (e.g. X-server, Web server, office applications,...). We select 'Choose manually' and select the following packages:

- 'apmd' is the Advanced Power Management Daemon. It is used for properly power off the computer after shut down.
- 'hdparm' is a tool for testing and configuring the hard disk parameters. In our case we did not had to manually set some settings here because the hard disk was already detected properly in the 'udma5' mode. We can check their speed with `hdparm -tT /dev/hda` or receive some disk information with `hdparm -i /dev/hda`.
- 'less' is a very useful console tool for displaying plaintext files.
- 'ssh' is the secure shell daemon and client of the OpenSSH project and is used as our remote shell.
- 'traceroute' is a network diagnostic tool for probing the routes of connections.

There are still some programs missing. But they are not the same on all machines. We are going to handle this in the following chapter 4 'Module Installation'.

3.3 Network Configuration

3.3.1 Hosts

In Debian GNU/Linux the network is configured through the */etc/network/interfaces* file. There we have to fill in the IP addresses, subnet masks, IP routes and the gateways.

Host Name	IP Address	Subnet Mask	Default Gateway
host1	10.1.0.100	255.255.255.0	10.1.0.10
host2	10.2.0.100	255.255.255.0	See Listing 3.1
host3	10.3.0.100	255.255.255.0	10.3.0.20

Table 3.2: IP addresses of the hosts

A separate DNS server is not needed in our small subnets. For the name resolution we provide the host names of our machines in the */etc/hosts* file (see Appendix B.2). On host1 and host3 one default route satisfies our network configuration. But on host2 the routes to the two other subnets have to be configured separately. Instead of one line e.g. 'gateway 10.1.0.10' we have to add the routes in */etc/network/interfaces* as follows:

```
up route add -net 10.3.0.0 netmask 255.255.255.0 gw 10.2.0.20 dev $IFACE
up route add -net 10.1.0.0 netmask 255.255.255.0 gw 10.2.0.10 dev $IFACE
```

Listing 3.1: /etc/network/interfaces of host2 (section of interest)

Because this host only has one network interface $IFACE is automatically substituted with eth0 during network initialisation.

3.3.2 Portal Server

This machine has build-in a PCI four-port network adapter. One port is meant for the connection to the university network and the Internet the three others for one subnet each. IP forwarding is not configured (*/proc/sys/net/ipv4/ip_forward* = 0) therefore the module portal machine is the only machine which can be directly connected from outside. Before we can configure the interfaces, we have to load the 'sundance' kernel module for the network adapter (`modprobe sundance`). The module name depends on the hardware you use and can be different with another network adapter. To make sure that it is loaded during every boot up, you have to add the module name in */etc/modules* (`echo "sundance" >> /etc/modules`). Again we set the IP addresses in the */etc/network/interfaces* file.

Interface	Gateway	Subnet mask	IP address
eth0	130.92.66.1	255.255.255.0	130.92.66.165 (gridlab06)
			130.92.66.166 (gridlab07)
eth1	-	255.255.255.0	10.1.0.1
eth2	-	255.255.255.0	10.2.0.1
eth3	-	255.255.255.0	10.3.0.1

Table 3.3: IP addresses of the portal server interfaces

3.4 Disable Direct Root Login

Because several administrative users might have the root password, we cannot distinguish with hindsight which person did which tasks with these rights. Therefore we disable the direct root login. Like that every administrator has to login with his personal user account first and switch with the `su` command to the root user. This is logged and can be traced down later when some security issues should arise. To disable root login on the terminals, change the */etc/pam.d/passwd* as follows:

```
# Disallows root logins except on tty's listed in /etc/securetty
# (Replaces the 'CONSOLE' setting from login.defs)
auth       requisite  pam_securetty.so
```

Listing 3.2: /etc/pam.d/passwd (section of interest)

Further you have to make sure that the */etc/securetty* file is empty. To disable root login over SSH set in the */etc/ssh/sshd_config*:

```
PermitRootLogin no
```

Listing 3.3: /etc/ssh/sshd_config (section of interest)

Chapter 4

Module Installation

This chapter describes the individual configurations that have done on the different machines to run the VITELS 'IP Security' module. The first part is a brief overview over the module elements, followed by a section about the host setup and a section about the portal server setup. At the end the module is, except of the router reset, ready to work. In this documentation only the installation and configuration of the module dependent scripts and program are discussed. This does not include the Web server and Web portal configuration.

4.1 Module Overview

The script layer of the module consists of different scripts for the following tasks:

Login Redirection - The student can access the machines and routers with his Web browser. Therefor a Java applet called Mindterm [11] is started on the student's machine which makes an SSH connection to the portal server. Depending on the machine the student wants to log in, the connection is made to a different user account. The portal server then redirects the login of this user to the appropriate host or router. If the student logs in on host1 the connection is made with the user host1 who gets redirected via SSH to the machine host1. It is the same procedure with the routers, but unlike the hosts, the redirection is managed over the serial terminal emulation Minicom [12].

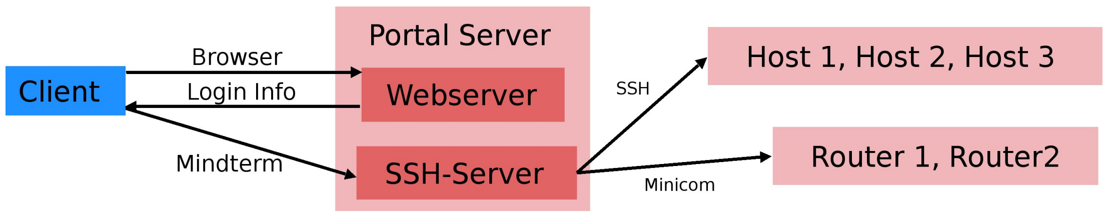

Figure 4.1: IP Security Module: Device Access Overview

Session Administration Script - For the authentication between Mindterm and the device users on the portal server, onetime passwords are generated by a script that is regularly run over a cronjob. It checks the current user in the reservation table and compares it with the lastly logged in user. If they do not match each other a new password is generated. Further it is responsible that the hosts and routers are in a clean state before a new user starts to solve the module.

Router Reset Script - It is a script that the user can access from the Web portal. It hard resets the routers to bring them back into the initial state. For more details check chapter 5 'Router Reset with help of a TFTP Server'.

4.2 Host Configuration

This section treats the specific configurations for the module on the machines host1 to host3.

4.2.1 User Configuration

On every machine (host1-3) a normal user for the remote login via SSH is needed. They are created with the following command (example for host1):

```
groupadd host1
useradd -g host1 -d /home/host1 -s /bin/bash -c 'IP Security Lab User' -m
host1
```

This has to be repeated on the other two hosts with the appropriate users host2 and host3.

4.2.2 SSH Configuration

The above created users do not log in via a normal password authentication, but via the public key authentication provided by the SSH protocol version 2. Make sure that it is properly configured in the */etc/ssh/sshd_config*:

```
RSAAuthentication       no
PubkeyAuthentication    yes
AuthorizedKeysFile      %h/.ssh/authorized_keys
```

Listing 4.1: /etc/ssh/sshd_config (section of interest)

For the public key authentication configuration take a look at the subsection 4.3.3 'Create Authentication Key Pair for SSH Remote Login'.

To make the SSH login a bit fancier and more informative than the Debian standard login, we change the */etc/motd* file to show the operating system and the current machine to the user:

```
  W E L C O M E   O N   H O S T 1

The operating system is Debian GNU/Linux
```

Listing 4.2: /etc/motd (from host1)

Because we do not need the information about the last login from the portal server, we turn it off in the */etc/ssh/sshd_config* file:

```
PrintLastLog            no
```

Listing 4.3: /etc/ssh/sshd_config (section of interest)

Do not forget to restart the SSH daemon after changes in the */etc/ssh/sshd_config*.

4.2.3 Additional Software

To use the hosts for network performance measuring and traffic sniffing we need specific software on certain machines.

NetPIPE
NetPIPE is used as bandwidth benchmarking tool. It is needed on the hosts 1 and 3:

```
apt-get install netpipe-tcp
```

TCPDUMP
For traffic sniffing we use TCPDUMP. Because it modifies the way how a network interface is working, it is able to transparently capture every packet passing the interface, it can normally only be invoked by the root user. To make it usable for normal users, we have to make it accessible from the normal user's execution path and set the setuid bit. This allows every user to run it with root privilege. It only has to be installed on host2:

```
apt-get install tcpdump
ln -s /usr/sbin/tcpdump /usr/bin/tcpdump
chmod u+s /usr/sbin/tcpdump
```

Telnet-Daemon
For establishing a telnet connection in the hands-on session, a telnet server is needed. 'telnetd' is the corresponding daemon which is started by inetd (Internet service daemon) [13]. inetd is a helper network daemon that starts the appropriate service when a connection to it is requested. In this case it starts telnetd when there is a connection request on port 23. On a Debian system inetd is installed and configured by default. On host1 and host3 we only have to run:

```
apt-get install telnetd
```

No manual configuration is needed here. The necessary line in the */etc/inetd.conf* file was already added by the telnetd installation procedure. The telnet client was installed with the basic Debian installation before.

4.3 Portal Server Configuration

This section describes how to configure the module portal machine. Because this machine is kind of a hub for all the services and users, the changes were more extensive compared to the other network hosts.

4.3.1 User Configuration

For every device, that we want to access over a console (router1, router2, host1, host2, host3), we have to add a user. To follow the exact commands see subsection 4.2.1 'User Configuration'.

4.3.2 SSH Configuration

Like on the other hosts, we also change the default message which appears after a successful login. Except the system administrator all the users, that see this message, will be module users who will be redirected to the appropriate device. We provide some useful information for them. The */etc/motd* is changed like this:

```
    W E L C O M E    T O    T H E    V I T E L S
   ___ ___     ___ ___ ___ _   _ ___ ___ _______   __
  |_ _| _ \___/ __| __/ __| | | | _ \_ _|_   _\ \ / /
   | ||  _/___\__ \ _| (__| |_| |   /| |  | |  \ V /
  |___|_|     |___/___\___|\___/|_|_\___| |_|   |_|

              M O D U L E

You're logged in on gridlab06.unibe.ch and you are going
to be redirected to the selected target device.

********************************************************
```

Listing 4.4: /etc/motd (from gridlab06)

Here is no need for information about the last login either. Again we prevent SSH from printing this message:

```
PrintLastLog            no
```

Listing 4.5: /etc/ssh/sshd_config (section of interest)

To provide the login via the Mindterm applet, you have to make sure that you enable 'PasswordAuthentication' in */etc/ssh/sshd_config* like this:

```
# Change to yes to enable tunnelled clear text passwords
PasswordAuthentication  yes
```

Listing 4.6: /etc/ssh/sshd_config (section of interest)

4.3.3 Create Authentication Key Pair for SSH Remote Login

It has already been said in the overview, that in case a certain user logs into the portal server he gets redirected to the appropriate device. For the host users this means that they get a shell of their host when logging in on the portal server. For a passwordless authentication between the portal server and the hosts, we create private-/public-key pairs. This has to be done on the portal server with all the affected users (host1-3, router1-2). For example run the following command with user host1 on gridlab06:

```
ssh-keygen -t dsa
```

When you get asked for a key name or a pass phrase, just press enter. We do not need a special input there. Afterwards we have to copy the content of the generated public key file *id_dsa.pub* into the */home/host1/.ssh/authorized_keys* file of host1. The private key is left on the portal server. This user now can log in from the portal server to his host without further authentication. Repeat this procedure for the three user-host combinations.

4.3.4 Login Redirection

To redirect the users to their corresponding device we change the default login shell */bin/bash* to a redirection script. This is done in the */etc/passwd* file. For changes in this critical file it is recommended to use the `vipw` command. It uses the default editor given by the $EDITOR system variable and performs a syntax check before saving and closing the *passwd* file.

```
host1:x:1011:1002:IP Security Lab User:/home/host1:/bin/host1
host2:x:1012:1003:IP Security Lab User:/home/host2:/bin/host2
host3:x:1013:1004:IP Security Lab User:/home/host3:/bin/host3
router1:x:1014:1005:IP Security Lab User:/home/router1:/bin/router1
router2:x:1015:1006:IP Security Lab User:/home/router2:/bin/router2
```

Listing 4.7: /etc/passwd (section of interest)

Now the login shells are */bin/hostX* or */bin/routerX*. These are scripts which open a new SSH or Minicom connection to the targeted device. For a host user it looks like this:

```
#!/bin/bash
# is called when user host1 logs into gridlab06.unibe.ch
# user is redirected to host1@host1 (needs authorized ssh keypair)

ssh host1
exit
```

Listing 4.8: /bin/host1

The router users are accordingly redirected to the routers. The occurring */root/namekill* script is going to be discussed in the next subsection:

```
#!/bin/bash
# filename: /bin/router1
# real tty connected to router1

# first: kill the existing jobs
sudo /root/namekill $USER minicom

# second: wait until minicom has shut down
sleep 6s

# third: start new minicom connection
/usr/bin/minicom -C /home/router1/minicom.log ttyS0
exit
```

Listing 4.9: /bin/router1

All the mentioned scripts have the permissions 755 what means that they are executable for all the users, but only writable by the root user.

4.3.5 The *namekill* Script

The small Perl script */root/namekill* (see Appendix C.4) kills specific processes of a user given as argument. This script is used because of the following reasons:

- Mindterm does not quit the SSH connections until it gets closed. Therefore a SSH connection with an open Minicom would remain established when you close the Mindterm window without exiting the login first. Because Minicom locks the serial terminal to the router we cannot access the still locked serial port anymore. As you can see in the */bin/routerX* script above, `namekill` is used to shut down the remaining Minicom jobs before it establishes a new connection.

- There is a session administration script */root/update_shadow_ldap.cnds.unibe.ch_ssl.pl* (see next subsection for details) that uses `namekill` for killing all the remaining jobs of a module user when he has logged out and finished the exercise on the module.

This script should only be executable by the user root. Like that we can control who else can execute it using sudo [14]. sudo allows us to give root permission to some users for executing specific programs while logging it. This is an advantage compared to the suid-bit where everybody can unrestrictedly run the privileged program. The installation is simple:

```
apt-get install sudo
```

To configure the permissions we have to edit the */etc/sudoers* file and add the following entries:

```
# the routers can kill their minicom sessions
router1 gridlab06=NOPASSWD: /root/namekill
router2 gridlab06=NOPASSWD: /root/namekill
```

Listing 4.10: /etc/sudoers (section of interest)

This gives the users 'router1' and 'router2' the right to execute */root/namekill* without password authentication from the current machine (in this case 'gridlab06').

4.3.6 The Session Administration Script

```
The script described in this subsection became obsolete after some Web portal changes
          in August 2006. It is only mentioned for the sake of completeness.
```

The script */root/update_shadow_ldap.cnds.unibe.ch_ssl.pl* (see Appendix C.5) is the main part of the module user management. Its permissions (700) only allow the user root to execute it. It checks the name of the student who has currently reserved a time slot for the module via an external LDAP server and writes it in the */etc/current_user* file. After every user change it generates with help of the password generator library *PwGenLib.pm* a new password for the Mindterm to portal server authentication and updates the password's hash in the */etc/shadow* file. To provide the password to the Web portal it is saved in plaintext in */etc/current_pw*. When a new module user now wants to log in, the Web portal looks for the password in */etc/current_pw* and includes it as parameter for the Mindterm applet.

The script is ran every minute by a cron job. Add in */etc/crontab*:

```
# check for current user every minute and update shadow if necessary
*  *  *  *  *   root    /root/update_shadow_ldap.cnds.unibe.ch_ssl.pl  \
>> /root/update_shadow.log 2>&1
```

Listing 4.12: /etc/crontab (section of interest)

To access the LDAP server, an additional Perl-LDAP module is used. Further it needs the mentioned files. 'dummy' is the default user when nobody is logged in on the portal:

```
apt-get install libnet-ldap-perl
echo "dummy" > /etc/current_name
touch /etc/current_pw
chmod 644 /etc/current_name /etc/current_pw
```

The mentioned *PwGenLib.pm* (see Appendix C.9) has to be in the same directory as the script. In our case this is */root*. It does not have to be executable so the permissions are 644.

4.3.7 Encrypted LDAP Querying

```
The script described in this subsection became obsolete after some Web portal changes
          in August 2006. It is only mentioned for the sake of completeness.
```

The session administration script connects to the LDAP server through an untrusted network. To protect the transmitted information, the connection should be encrypted. With Stunnel [15] we can wrap any plaintext connection into a SSL encrypted connection.

```
apt-get install stunnel
```

The *update_shadow_ldap.cnds.unibe.ch_ssl.pl* script makes the LDAP connection to localhost port 636. Stunnel encrypts and forwards the data it receives on port 636 to the remote LDAP server. To setup the secure tunnel we need another small script called */usr/local/bin/stunnel_ldap.sh*:

```
#!/bin/bash
# redirects and encrypts LDAP connection on localhost to LDAP server

/usr/sbin/stunnel -c -d 636 -r ldap.cnds.unibe.ch:636
```

Listing 4.14: /usr/local/bin/stunnel_ldap.sh

It is initialised by cron with the following entry in the */etc/crontab*:

```
@reboot         root    /usr/local/bin/stunnel_ldap.sh
```

Listing 4.15: /etc/crontab (section of interest)

4.3.8 Module Reset Script

For resetting the hosts and the routers if the user of the module changes, a small script is evoked from *update_shadow_ldap.cnds.unibe.ch_ssl.pl*. It deletes the log files in the user's home directories on the three hosts and resets the routers. The script called *module_reset.sh* is placed with permissions 744 in the */root* directory:

```
#!/bin/bash
# script to reset the hosts and routers
# 29.11.2005 by reto gantenbein <gantenbe@iam.unibe.ch>

# empty the home directories of the hosts
for USER in "host1" "host2" "host3"
do
  sudo -u $USER ssh $USER@$USER rm -rf /home/$USER/*
  sudo -u $USER ssh $USER@$USER rm /home/$USER/.bash_history
done

# reset routers
for ROUTER in "router1" "router2"
do
  /var/php_safe_mode_bin/pw_reset.pl $ROUTER
done

exit 0
```

Listing 4.16: /root/module_reset.sh

4.3.9 Serial Port Configuration

The module portal machine has a total of three serial ports. Two of them are used for the router consoles and one for switching the relay on which the router's power supplies are connected.

Debian has a package called 'setserial' that is used to set the configuration information associated with the serial port including which I/O-port and what IRQ a particular serial port allocates. Install it with:

```
apt-get install setserial
```

Its configuration file is */etc/serial.conf*. It contains the assignments of the hardware serial ports to the serial terminal devices (ttyS). If you do not know the IRQs and ports of the connectors, keep your eyes open on the system startup or consult them with:

```
dmesg | grep ttyS
```

At the end the file should look similar to this:

```
# /etc/serial.conf
# serial port configuration file
/dev/ttyS0 uart 16550A port 0x3f8 irq 4 baud_base 115200 spd_normal skip_test
/dev/ttyS1 uart 16550A port 0x8400 irq 10 baud_base 115200 spd_normal skip_test
/dev/ttyS2 uart 16550A port 0x8000 irq 10 baud_base 115200 spd_normal skip_test
```

Listing 4.17: /etc/serial.conf

To assign this information on every boot up, 'setserial' has to be added to the system initialisation:

```
ln -s ../init.d/setserial /etc/rc2.d/S12setserial
```

4.3.10 Minicom Configuration

As stated in the overview (section 4.1) the redirection to the hosts is done using SSH. For the routers we need a different approach because their network interfaces are not configured at the beginning. We have to access them with the terminal emulation program Minicom which connects to the routers over the serial line:

```
apt-get install minicom
```

The connections can be configured by starting Minicom with the `-s` option. A menu with all required parameters will appear. The configuration of each device should be saved in a separate configuration file. Its name depends on the terminal it is used for (e.g. */etc/minicom/minirc.ttyS0* for */dev/ttyS0*) and should look like this:

```
# Machine-generated file - use "minicom -s" to change parameters.
pr port          /dev/ttyS0
pu baudrate      9600
pu bits          8
pu parity        N
pu stopbits      1
pu scriptprog    /usr/bin/runscript
pu minit         ~^M~
pu mreset
pu updir         /tftpboot
```

Listing 4.18: /etc/minicom/minirc.ttyS0

Because the router users (router1, router2) use the serial port terminals (ttyS0 and ttyS1), which are owned by the 'dialout' group, they have to be added to it. This changes were made in the */etc/group* file. We can use the command `vigr` that works similar like the previously discussed `vipw`:

```
dialout:x:20:router1,router2
```

Listing 4.19: /etc/group (section of interest)

4.3.11 Mail System Configuration

A decent linux installation should also have a running mail system. While our laboratory hosts are not connected to a mail server and no installed service really needs a mail system, that looks different on our portal server. Especially for monitoring and alerting purpose, it is important that a service is able to send mails. On the other hand it is not really necessary that our server can also receive mails. Therefore the preinstalled mail transfer agent Exim [16] is overkill for our needs. We can remove it and install ssmtp [17] instead.

```
apt-get remove exim4 && apt-get install ssmtp
```

The advantage of ssmtp is the small configuration file (*/etc/ssmtp/ssmtp.conf*). It is very clear and quickly filled with the appropriate values. First we have to enter an email address which will receives the system mails to the root user. The other important value is the mailhub variable. Inside the university network ubecx.unibe.ch can be used. The other values are printed in the configuration file below.

```
#
# Config file for sSMTP sendmail
#
# The person who gets all mail for userids < 1000
# Make this empty to disable rewriting.
root=gantenbe@iam.unibe.ch

# The place where the mail goes. The actual machine name is required no
# MX records are consulted. Commonly mailhosts are named mail.domain.com
mailhub=ubecx.unibe.ch
```

```
# Where will the mail seem to come from?
rewriteDomain=unibe.ch

# The full hostname
hostname=gridlab07.unibe.ch

# Are users allowed to set their own From: address?
# YES - Allow the user to specify their own From: address
# NO - Use the system generated From: address
FromLineOverride=YES
```

Listing 4.20: /etc/ssmtp/ssmtp.conf

Another advantage of ssmtp is that it can be invoked like the widely used mail transfer agent Sendmail [18]. To use ssmtp with the normal `mail` command we have to delete the symlink */usr/sbin/sendmail* -> *exim4* and replace it with a corresponding symlink to the ssmtp binary.

```
rm /usr/sbin/sendmail
ln -s /usr/sbin/ssmtp /usr/sbin/sendmail
```

4.4 Testing the Setup

4.4.1 Manual Testing

The most important parts of our module should work now. The login redirection can be tested by running `su host1` or `su router1` on the module portal machine. Therewith we execute the script */bin/host1* or */bin/router1* and we should get a terminal of the corresponding device.

The session administration script cannot be tested independently of the Web portal. To check its functions, reserve a time slot in the module scheduling system and then try to log in on the Web portal. If it does not work instantly check the log files described in the section 7.2 'Troubleshooting Problems'.

4.4.2 Monitoring Script

For monitoring the basic module functionality, there is a small bash script. It checks if the hosts are still alive, if the Web server and the SSH daemon is still running and accepting connections. With the ALERT_MAIL and CC_MAIL variables at the beginning you can specify who receives an alert mail if one of the mentioned hosts/services is not reachable.

```
#!/bin/bash

# Script to test basic 'IP Security' module services
# Reto Gantenbein (gantenbe@iam.unibe.ch)

LOGFILE=/root/module_status.log
```

```
ALERT_MAIL=gantenbe@iam.unibe.ch
CC_MAIL=bernoull@iam.unibe.ch

#
# Test Hosts
#
for HOST in host1 host2 host3 ; do
  ping -c1 $HOST >> /dev/null 2>&1
  if [ $? -eq 1 ] ; then
    echo "$HOST does not respond!" >> $LOGFILE
  fi
done

#
# Test Apache2
#
APACHE_PID=$(pidof apache2)
APACHE_PORT=$(netstat -napt | grep 0.0.0.0:443)

if [ -z "$APACHE_PID" -o -z "$APACHE_PORT" ] ; then
  echo "Apache2 is not running properly!" >> $LOGFILE
fi

#
# Test SSH
#
SSH_PID=$(pidof sshd)
SSH_PORT=$(netstat -napt | grep 0.0.0.0:22)

if [ -z "$SSH_PID" -o -z "$SSH_PORT" ] ; then
  echo "SSH is not running properly!" >> $LOGFILE
fi

#
# Send alert mail
#
if [ -f $LOGFILE ] ; then
   mail -s "$HOSTNAME.unibe.ch: IP-Security Module Alert" -c $CC_MAIL $ALERT_MAIL \
   < $LOGFILE
   rm $LOGFILE
fi
```

Listing 4.21: /root/module_test.sh

To let it run every hour, a symlink in */etc/cron.hourly/* is created:

```
ln -s ../../root/module_test.sh /etc/cron.hourly/module_test.sh
```

It is only a small script for very basic monitoring. Unfortunately the SSH connections to the hosts and the Minicom connection are not tested. But it is already quite helpful to be notified if something is not working properly.

4.4.3 Logwatch

Finally also the general operating system functionality should be monitored. That can be done by Logwatch [19]. It is a Perl script for monitoring the system log files. It can send a daily report of the happenings on the system to the administrator. The installation is simple like always:

```
apt-get install logwatch
```

Its configuration */etc/logwatch/conf/logwatch.conf* does not need any changes to produce meaningful reports. For generating a first report Logwatch can be invoked with the root user with `logwatch`. For receiving daily reports, Logwatch starts itself with the small script */etc/cron.daily/00logwatch* which is installed by default.

Chapter 5

Router Reset with help of a TFTP Server

5.1 Overview

To configure the routers, the students need full administrator permissions. We can activate the privileged mode by typing `enable` on the router console. On a productive router this would normally be protected by a password authentication. With the gained rights people have the possibility to set an administrator password, to delete files on the router flash memory, to change configuration registers and much more. Without proper reset this can lead to an inoperative router for the following user. Therefore a solution was created to reset the routers to an initial state. They will be hard reset by a power switching relay card and all original configurations have to be restored during system startup. The original producer of the 'IP Security' module Stefan Zimmerli wrote a script and two helper libraries for this task. There is the *RelaisLib.pm* (see Appendix C.10) which is responsible for controlling the serial line relay. The *RouterLib.pm* (see Appendix C.11) is used for sending commands to the router console and for locking the routers during the reset. The actual reset script, which is accessed by the Web portal, is the *pw_reset.pl* (see Appendix C.12) and coordinates the router reset steps. It initiates the locking, the power supply switching and the configuration steps to prepare the router for next use.

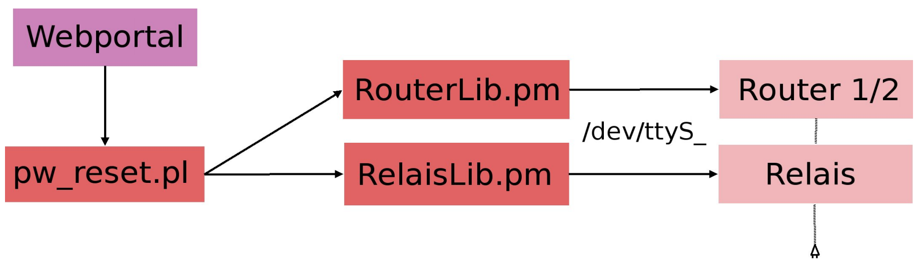

Figure 5.1: Reset Activation Overview

With the approach of Stefan Zimmerli the routers are lost when somebody intentionally or unintentionally deletes the router's IOS (Cisco's Router Operating System). In such a case the module maintainer has to manually restore the IOS on the router with help of a PCMCIA flash card and a backup copy of the image. Another disadvantage of his approach is that the students first have to perform some manual resetting tasks to ensure a clean environment. Obviously, this is not a very comfortable solution for the module maintainer and the students.

Therefore a new approach should improve the situation. Fortunately there is an easy solution. On every router reset a fresh IOS image that overwrites the entire router memory can be loaded from a TFTP server. Therewith it is also possible to choose from different IOS versions with variable features. However this solution can only be implemented on one of our two modules because only the ROMmon (for explanation see section 5.3 'Router Reset in Detail') of the Cisco 2600 routers support an instant loading of an IOS over the network.

5.2 What is TFTP?

TFTP (Trivial File Transfer Protocol) was first defined in 1980 and is as the name suggests a very simple file transfer protocol. Because of its simplicity, it is first of all used for booting diskless computer systems from centralised operating system image sources. Its specification can be found in the RFC 1350.

The transport protocol TFTP uses UDP on the server side port 69. The only possible messages that can be sent with TFTP are RRQ (read request), WRQ (write request), ACK (acknowledgement) and DATA (data packets). Therefore the possibilities are limited to read from or write to a remote server. In comparison to FTP, TFTP does not support directory listing, authentication, compression or encryption mechanisms. Further the maximal file size cannot exceed 32 MB.

TFTP File Transfer (example):

1. The initiating host A sends a RRQ to the server B containing the filename and the transfer mode (ASCII or binary).

2. The server B answers with an ACK and the requested file is delivered within DATA packets. The destination host A confirms every received DATA packet with a numbered ACK.

3. B sends a last DATA packet containing less than a full size block of data to signal the end of file. If the file size is an exact multiple of the block size, the last DATA packet contains 0 bytes of data.

5.3 Router Reset in Detail

To understand the detailed reset cycle and the upcoming script changes, the steps of the resetting task of the *pw_reset.pl* script, with help of the *RouterLib.pm*, will be explained now:

1. A command to interrupt the power connection to the routers is sent to the relay card. After a short moment the power supply is switched on again.

2. The routers first try to load a basic command line system called ROMmon (ROM monitor). Before it starts to initialise the IOS, the boot up is interrupted by a BREAK signal (pulse_break_on). Therewith we can force the router to stay in the ROMmon mode.

3. In the ROMmon we set some register values by sending `confreg 0x2142`. This means, that the router should boot the default ROM software when he cannot boot from the network (0x2000). It should not be possible to interrupt the next boot up by sending a BREAK signal (0x0100). The IOS initialisation should not take care of the earlier created configuration files (0x0040). The last value 0x0002 is only used in the Cisco 3600 setup and means that the ROMmon should start the IOS image previously defined with the `boot system` command.

4. The routers are reset again and start with the new register settings. On the Cisco 3600 setup the IOS is loaded now. The Cisco 2600 routers go into ROMmon mode again because we did not set an IOS to load. There the *RouterLib.pm* then configures a network interface and a fresh IOS image is loaded from the TFTP server. The image overwrites the whole router memory. So we do not have to care about configuration files of the previous user. After the new IOS is initialised the Cisco 2600 routers are ready to use by the module user.

5. The running IOS on the Cisco 3600 routers still cannot be used after boot up. We first have to ensure that old configuration files cannot be accessed by a new user. Therefore we overwrite the startup configuration 'startup-config', that was created by the previous user, by the yet unconfigured system state accessible under 'running-config'. Only after a next reload we can give console access to the module user.

5.4 Reset Script Changes

The original script by Stefan Zimmerli uses for both router types the same complicated approach described in the previous section for the Cisco 3600 routers. To give an impression how this is coded, the corresponding section is listed below:

```
$myport->pulse_break_on(1000);

send_router_command($myport,"confreg 0x2142\n",1);
send_router_command($myport,"reset\n",106);

send_router_command($myport,"no\n",2);
send_router_command($myport,"\r",2);
send_router_command($myport,"\r",2);

send_router_command($myport,"enable\n",1);

send_router_command($myport,"configure\n",1);
send_router_command($myport,"terminal\n",1);
send_router_command($myport,"config-register 0x2102\n",1);
send_router_command($myport,"exit\n",1);
send_router_command($myport,"copy running-config startup-config\n",1);
send_router_command($myport,"startup-config\n",12);
```

```
    send_router_command($myport,"reload\n",1);
    send_router_command($myport,"\n",106);
    send_router_command($myport,"\r",2);
```

Listing 5.1: /usr/share/perl5/RouterLib.pm (Cisco 3600 reset)

The new setup realised with the Cisco 2600 routers, now retrieves a file called *currentIOS* over the ROMmon TFTP client `tftpdnld`. *currentIOS* is a symlink in the */tftpboot* folder of the portal server pointing to an applicable image. Make sure that this symlink always provides a matching IOS file.

```
    $myport->pulse_break_on(1000);

    send_router_command($myport,"confreg 0x2140\n",1);
    send_router_command($myport,"reset\n",10);

    if ($myrouter eq $RouterLib::ROUTER1) {
        send_router_command($myport,"IP_ADDRESS=10.2.0.10\n",1);
    } else {
        send_router_command($myport,"IP_ADDRESS=10.2.0.20\n",1);
    }

    send_router_command($myport,"IP_SUBNET_MASK=255.255.255.0\n",1);
    send_router_command($myport,"DEFAULT_GATEWAY=10.2.0.1\n",1);
    send_router_command($myport,"TFTP_SERVER=10.2.0.1\n",1);
    send_router_command($myport,"TFTP_FILE=currentIOS\n",1);
    send_router_command($myport,"tftpdnld\n",2);
    send_router_command($myport,"yes\n",170);
    send_router_command($myport,"boot flash:currentIOS\n",80);
    send_router_command($myport,"no\n",1);
```

Listing 5.2: /usr/share/perl5/RouterLib.pm (Cisco 2600 reset)

We also thought about changing the Cisco 3600 bootstrapping to make it less error-prone. Unfortunately their ROMmon version only support an IOS restore from an external data source over the serial line. It uses the XMODEM [20] file transfer protocol. Such an approach is not suitable for our use because the small bandwidth of the serial line delays a restore about three-quarters of an hour (experienced value).

5.5 Installing the Router Reset Script

In chapter 5.3 we already saw how the reset mechanism works, now we have to put the scripts into the right places. The basic libraries, *RouterLib.pm* and *RelaisLib.pm*, are found in */usr/share/perl5* which should be in the path of the Perl interpreter. They need the permissions 644. Because the *RelaisLib.pm* requires the Perl extension Device:SerialPort, it must be installed with:

```
apt-get install libdevice-serial-port
```

For security reasons the PHP interpreter of the Web server should normally not be allowed to execute normal programs on the machine. Therefore we create as user root a seperate directory with the needed programs that have to be accessed by the Web site.

```
mkdir /var/php_safe_mode_bin
```

First of all the *pw_reset.pl* script is placed in there. It should not be allowed to reset the routers for every user therefore the permissions are 700 with owner root. Via sudo we give the Web server the necessary permissions to execute the script. Once more we have to edit the */etc/sudoers* file. The attached line means that the user 'www-data', which is the user the Web server runs as, can execute */var/php_safe_mode_bin/pw_reset.pl* from the local machine (in this case 'gridlab06') without password authentication:

```
# Apache user (www-data) can execute pw_reset.pl without authentication
www-data    gridlab07=NOPASSWD: /var/php_safe_mode_bin/pw_reset.pl
```

Listing 5.3: /etc/sudoers (section of interest)

5.6 TFTP Server Configuration

The portal server has to run a TFTP file server where the routers can get their IOS images from. For this task we install the very powerful and flexible TFTP daemon [21] from the OpenBSD project. In Debian the concerning package is called 'tftpd-hpa'. For testing purpose also a tftp client is installed.

```
apt-get install tftp tftpd-hpa
```

Such as the Telnet service the TFTP daemon can be started over the inet daemon. Therefore the following line has to be added in the */etc/inetd.conf*:

```
#:BOOT: Tftp service is provided primarily for booting. Most sites
# run this only on machines acting as "boot servers."
tftp  dgram  udp  wait  root  /usr/sbin/tcpd  /usr/sbin/in.tftpd \
-u tftpd -s /tftpboot -a 10.2.0.1:69
```

Listing 5.4: /etc/inetd.conf (section of interest)

The `-u <username>` argument specifies the user name which the service will run as. For security reasons we create a special user 'tftpd':

```
useradd -g users -d /tftpboot -s /bin/false tftpd
```

The `-s <directory>` argument sets the file directory to */tftpboot* which requires reading and writing permissions for everybody:

```
mkdir /tftpboot && chmod 777 /tftpboot
chown tftpd:users /tftpboot
```

Now we only have to store our router IOS in the */tftpboot* folder. Like mentioned before the Cisco 2600 routers are configured that they load an IOS called *currentIOS*. It is possible to store different IOS versions into this folder and just create a symlink called *currentIOS* to the desired IOS version. Maybe it would be eligible to have the possibility to change this symlink from an interface of the coming version of the Web portal. Like that different IOS versions could be used at the same time. Our TFTP daemon is also able to receive files. If you want to enable this feature you have to add `-c` in the command line of the */etc/inetd.conf*.

5.7 Manual Restore of a Router IOS over a TFTP Server

Unfortunately the Cisco 3600 routers do not provide a direct access to a TFTP server from the ROMmon mode. We can only download a new IOS image over TFTP within an already running IOS. The following steps have to be followed to do so:

```
Router# enable
Router# show flash:
Directory of flash:

  1   -rw-   13843376   <no date>   c3620-jk9s-mz.122-23e.bin

16777216 bytes total (2933776 bytes free)
Router# delete flash:c3620-jk9s-mz.122-23e.bin
Router# copy tftp://10.1.0.1/c3620-js56i-mz.121-5.T9.bin flash:
Destination filename [c3620-js56i-mz.121-5.T9.bin]?
Accessing tftp://10.1.0.1/c3620-js56i-mz.121-5.T9.bin...
Erase flash:  before copying?  [confirm]
Erasing the flash filesystem will remove all files!  Continue?  [confirm]
Erasing device...  eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
Erase of flash:  complete
Loading c3620-js56i-mz.121-5.T9.bin from 10.1.0.1 (via FastEthernet0/1):  !!!
[OK - 13063896 bytes]

Verifying checksum...  OK (0x704F)
13063896 bytes copied in 117.912 secs (110794 bytes/sec)
Router# reload
```

Chapter 6

Module Content

For solving the hands-on session a modern Web browser (e.g. Firefox, Safari) with a working Java support is required. Also the Web site certificate signed by the SWITCH certificate authority [2] has to be (temporary) accepted. Because there are only a limited number of laboratory environments, every student first has to reserve a time slot in which he has access to the laboratory network. During his time slot, he can enter the portal site of the module and therein open Java Web applet terminals to the devices. Fig. 6.1 shows the laboratory network topology:

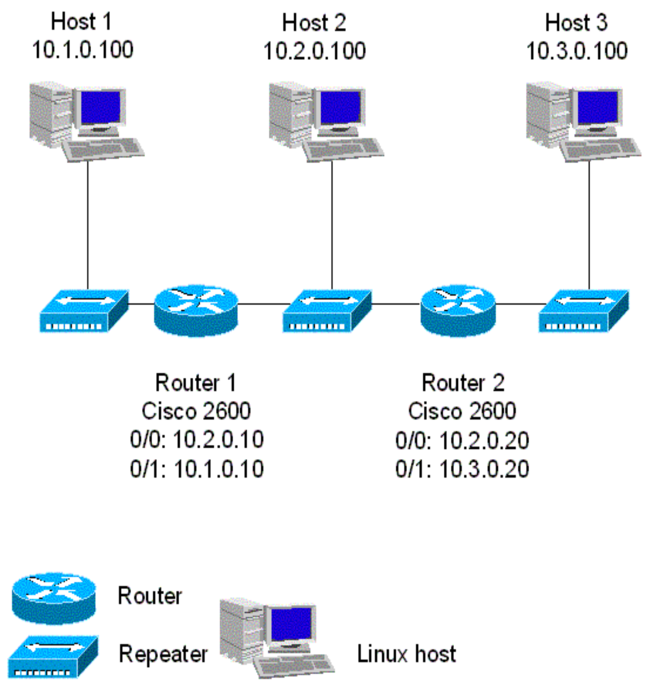

Figure 6.1: Laboratory Network Topology

We will go now trough every exercise of this hands-on session with a detailed explanation. Before you can start with configuring the routers, you have to make sure, according to the given instructions, that they are in a fully reset state. On the module which has the Cisco 2600 routers (IPSec01) we have implemented a new way of resetting the routers where this step is dispensable (see chapter 5). We will follow exactly the given instructions given by the hands-on session Web site.

6.1 Hands-On Session

6.1.1 Basic Router Configuration

Instruction: *"First of all log into both Cisco routers and erase the current config."*

```
Router>enable
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#erase startup-config
Router(config)#reload
```

With `enable` we enter the privileged mode of the router. In a normal environment you have to provide a password at this point, but not with this laboratory routers. `configure terminal` then changes the prompt from query to configuration mode. After erasing the configuration, we reload the router to make sure that it is in a clean state.

Instruction: *"The next step is to give the routers a host name and to configure their interfaces identically as shown in the laboratory configuration."*

The `interface <if-name>` command selects the interface for which the following configuration commands are applied. You can exit this mode again with `exit`. The identifiers of the interfaces may vary from router to router depending on the used network devices. They can be showed by running `show interfaces` in the query mode.

```
Router(config)#hostname router1
router1(config)#interface FastEthernet0/1
router1(config-if)#ip address 10.1.0.10 255.255.255.0
router1(config-if)#no shutdown
router1(config-if)#exit
9w1d:  %LINK-3-UPDOWN: Interface FastEthernet0/1, changed state to up
9w1d:  %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0,
changed up
router1(config)#interface FastEthernet0/0
router1(config-if)#ip address 10.2.0.10 255.255.255.0
router1(config-if)#no shutdown
```

```
router1(config-if)#exit
9w1d:  %LINK-3-UPDOWN: Interface FastEthernet0/0, changed state to up
9w1d:  %LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/1,
changed up
```

Now we already configured and activated both interfaces on the router.

6.1.2 Setting up Routing Information Protocol (RIP)

Instruction: *"After you have configured the interfaces, set up the IP routing. Use RIP version 2 for routing."*

```
router1(config)#router rip
router1(config-router)#network 10.2.0.0
router1(config-router)#neighbor 10.2.0.20
router1(config-router)#version 2
router1(config-router)#exit
```

`router rip` enables a RIP routing process which changes the prompt into the routing configuration mode. With `network <ip-network>` we associate the network where the RIP routing is active, and with `neighbor <ip>` we define the unicast receiver of our routing information. In this case it is the second router. Normally RIP would be a broadcast protocol.

The same steps have to be performed with adopted IP addresses on the second router. Now every network device should be reachable by any other network device. We will test this in the next step.

6.1.3 Testing RIP

Instruction: *"Ping every host in your net, from host to host and from router to host"*

Here is an example output:

```
host1@host1:~$ ping -c3 host2
PING host2 (10.2.0.100) 56(84) bytes of data.
64 bytes from host2 (10.2.0.100):  icmp_seq=1 ttl=63 time=0.719 ms
64 bytes from host2 (10.2.0.100):  icmp_seq=2 ttl=63 time=0.733 ms
64 bytes from host2 (10.2.0.100):  icmp_seq=3 ttl=63 time=0.695 ms
--- host2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.695/0.715/0.733/0.034 ms
```

Instruction: *"Retry but use the command 'debug ip packet' first."*

```
router1#debug ip packet
IP packet debugging is on
5w4d:  IP: s=10.1.0.100 (FastEthernet0/1), d=10.3.0.100 (FastEthernet0/0),
g=10.2.0.20, len 84, forward
```

After switching on the IP packet debugging on the router, all passing IP packets are being listed on the router console. The above listing shows an ICMP ping packet from host1 to host3. We can see the source IP address of the packet, the source interface on the router, the destination IP address of the packet, the destination interface on the router, the next hop IP address, the packet size and the packet rule (in this case 'forward'). To switch the debugging off again, the previously given command has to be prefixed with `no`. Most features on Cisco routers can be disabled again like this:

```
router1#no debug ip packet
IP packet debugging is off
```

Instruction: *"Use traceroute to examine your connections."*

```
host1@host1:~$ traceroute host2
traceroute to host2 (10.2.0.100), 30 hops max, 38 byte packets
1 router1_0 (10.1.0.10) 2.089 ms 1.025 ms 1.025 ms
2 host2 (10.2.0.100) 0.791 ms 0.718 ms 0.722 ms

host1@host1:~$ traceroute host3
traceroute to host3 (10.3.0.100), 30 hops max, 38 byte packets
1 router1_0 (10.1.0.10) 1.013 ms 1.074 ms 0.956 ms
2 router2_0 (10.2.0.20) 2.381 ms 1.458 ms 1.334 ms
3 host3 (10.3.0.100) 1.974 ms 1.169 ms 1.133 ms
```

The packets from host1 to host2 only pass one router while the packets to host3 have to pass both routers. Now we can be sure that all traffic successfully arrives at the desired host.

Instruction: *"Use NetPIPE (the command is: 'NPtcp') to measure the bandwidth from Host 1 to Host 3"*

NetPIPE [3] is used on host1 and host3. On the target machine it has to be started as sink with no argument and on the sender machine as traffic generator with the target host as argument. With `-o <filename>` we can generate a log file of the measuring for future reference.

```
host3@host3:~$ NPtcp
Send and receive buffers are 16384 and 87380 bytes
(A bug in Linux doubles the requested buffer sizes)
```

```
host1@host1:∼$ NPtcp -h host3 -o netpipe1.stats
Sending output to netpipe1.stats
Send and receive buffers are 16384 and 87380 bytes
(A bug in Linux doubles the requested buffer sizes)
Now starting the main loop
0:  1 bytes 203 times --> 0.02 Mbps in 479.03 usec
1:  2 bytes 208 times --> 0.03 Mbps in 477.51 usec
2:  3 bytes 209 times --> 0.05 Mbps in 479.06 usec
...
72:  24573 bytes 3 times --> 7.07 Mbps in 26533.16 usec
73:  24579 bytes 3 times --> 7.11 Mbps in 26358.85 usec
74:  32765 bytes 3 times --> 6.79 Mbps in 36839.49 usec
```

The bandwidth settles down at around 7 Mbps. The reason of this nowadays low value are the limiting Ethernet hubs (10 Mbps) which are installed between the routers and the hosts. The absolute value is not really important here. We only need it for further comparison in a later exercise.

Instruction: *"Make a telnet session between Host 1 and Host 3 and try to sniff the password using TCPDUMP on Host 2."*

Because the network devices are connected through hubs, host2 can receive the whole network traffic. A hub always forwards an incoming frame to all ports, except the incoming. This means that all the traffic sent between the routers is also received by host2. There TCPDUMP [4] can be used to log the interesting traffic between host1 and host3 which send their traffic also between the routers. It is an easy task to sniff a password from telnet because all data, also the password, is sent in plain text. To convince ourselves of this large security issue we will visualise these interesting packets now. First we set a filter to only log the needed packets sent from host3 to host1 and on port 23. Then we start TCPDUMP on host2. With the argument `-F <filename>` we specify the previously defined filter and with `-w <filename>` we create again a capture file. Finally we can try to log in from host3 to host1 via telnet. You will not achieve to successfully log in on host1 because the password is unknown. But the latter is also not the aim of this exercise. We are only interested in what host2 receives when we enter the imaginary password.

```
host2@host2:∼$ echo 'port 23 and src host 10.3.0.100 and
dst host 10.1.0.100' > tcpdump.filter

host2@host2:∼$ tcpdump -F tcpdump.filter -w tcpdump1.log
tcpdump:  listening on eth0, link-type EN10MB (Ethernet), capture size 96
bytes
44 packets captured
44 packets received by filter
0 packets dropped by kernel
```

```
host3@host3:~$ telnet host1
Trying 10.1.0.100...
Connected to host1.
Escape character is 'ˆ]'.
Debian GNU/Linux 3.1 host1
host1 login:  host1
Password:  reto
Login incorrect
```

The log file with the captured packets can also be analyzed with TCPDUMP. The option `-XX` prints the entire packet, including the link level header, in ASCII and hexadecimal notation. Thanks to the filter, it is not a heavy task anymore to identify the packets containing the password. To have a closer look at them they are printed below.

```
host2@host2:~$ tcpdump -r tcpdump1.log -XX
...
00:16:44.955352 IP host3.33180 > host1.telnet:  P 83:84(1) ack 110 win 5840
<nop,nop,timestamp 1840889587 1029094214>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 7272 4000 3f06 b475 0a03 0064 0a01 .5rr@.?..u...d..
   0x0020:  0064 819c 0017 744f ecd4 7dc7 040c 8018 .d....tO..}.....
   0x0030:  16d0 5923 0000 0101 080a 6db9 bef3 3d56 ..Y#......m...=V
   0x0040:  bb46 72                                 .Fr
00:16:45.043059 IP host3.33180 > host1.telnet:  P 84:85(1) ack 110 win 5840
<nop,nop,timestamp 1840889675 1029095076>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 7273 4000 3f06 b474 0a03 0064 0a01 .5rs@.?..t...d..
   0x0020:  0064 819c 0017 744f ecd5 7dc7 040c 8018 .d....tO..}.....
   0x0030:  16d0 4e6c 0000 0101 080a 6db9 bf4b 3d56 ..Nl......m..K=V
   0x0040:  bea4 65                                 ..e
00:16:45.115746 IP host3.33180 > host1.telnet:  P 85:86(1) ack 110 win 5840
<nop,nop,timestamp 1840889748 1029095123>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 7274 4000 3f06 b473 0a03 0064 0a01 .5rt@.?..s...d..
   0x0020:  0064 819c 0017 744f ecd6 7dc7 040c 8018 .d....tO..}.....
   0x0030:  16d0 49f3 0000 0101 080a 6db9 bf94 3d56 ..I.......m...=V
   0x0040:  bed3 74                                 ..t
00:16:45.251787 IP host3.33180 > host1.telnet:  P 86:87(1) ack 110 win 5840
<nop,nop,timestamp 1840889884 1029095196>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 7275 4000 3f06 b472 0a03 0064 0a01 .5ru@.?..r...d..
   0x0020:  0064 819c 0017 744f ecd7 7dc7 040c 8018 .d....tO..}.....
   0x0030:  16d0 4821 0000 0101 080a 6db9 c01c 3d56 ..H!......m...=V
   0x0040:  bf1c 6f                                 ..o
```

Let us have a closer look at the first packet. We see it fully printed in hexadecimal and ASCII notation. Now we have to remember how a network packet is assembled. At the beginning there is a 14 bytes long Ethernet header. The first 48 bits are the destination MAC address (00:d0:58:65:f5:60), followed by 48 bits of the source MAC address (00:d0:58:65:f5:40). They are nearly the same because they both belong to Cisco routers. The next 16 bits (0x0800) name the used layer 2 protocol which is in this case IPv4. Then there are 20 bytes of the Internet Protocol header. It is split up into 4 bits (0x4) version information (version 4), 4 bits (0x5) header size information (20 bytes), 8 bits (0x10) type of service, 16 bits (0x0035) total package length (53 bytes), 16 bits (0x7272) identification number (29298), 16 bits (0x4000) fragmentation information, 8 bits (0x3f) time to live value (63), 8 bits (0x06) layer 3 protocol information (TCP) and a 16-bit header checksum. As next there is a 32-byte TCP header. It consists of two times 16 bits source (0x0a030064 = 10.3.0.100) and destination (0x0a010064 = 10.1.0.100) IP addresses, two times 8 bits source (0x819c = 32356) and destination (0x0017 = 23) ports, 32 bits (0x744fecd4) sequence number, 32 bits (0x7dc7040c) acknowledgement number, 8 bits (0x80) header size information (32 bytes), 8 bits (0x18) control flags (ACK PSH), 16 bits (0x16d0) window size (5840), 16 bits (0x274c) checksum and 96 bits options. At the end follows the interesting part namely the 8-bit (0x72) payload. Translated into ASCII notation it means 'r', the first character of the password. When concerning the other packets, we finally see that every character of the password is separately transmitted in plaintext in the payload of a TCP packet.

6.1.4 Setting up the VPN

We tested the entire installation with no security options enabled. This leads to a quite unsafe environment when using plaintext protocols (e.g. Telnet, FTP, HTTP). We also saw that the traffic rate limitation is caused by the unintelligent network equipment (Ethernet hubs). Now we will enable an encrypted connection between the two routers and will have a look what happens when we repeat our previous tests. This means we setup a "safe" virtual private network between the networks 10.1.0.0 and 10.3.0.0. All traffic through the insecure 10.2.0.0 network will be encrypted. Again we follow the instructions in the VITELS module:

Instruction: *"Create DSS keys on the routers."*

On both routers a DSS key has to be generated with the `crypto key generate dss <keyname>` command. Here they are called cisco2600-1 on router1 and cisco2600-2 on router2.

```
router1(config)#crypto key generate dss cisco2600-1
Generating DSS keys ....
 [OK]

router2(config)#crypto key generate dss cisco2600-2
```

```
Generating DSS keys ....
 [OK]
```

Instruction: *"Exchange the DSS keys."*

With the `crypto key exchange dss` command we can exchange our previously generated DSS keys. Therefor one router has to be assigned to have the passive role what means that it is listening for key exchange requests. The other router afterwards initiates the key exchange.

```
router2(config)#crypto key exchange dss passive
Enter escape character to abort if connection does not complete.
Wait for connection from peer[confirm]
Waiting ....

Public key for cisco2600-1:
 Serial Number 593CA972
 Fingerprint   C9F6 7255 FF19 D7BD D8BD

Add this public key to the configuration?  [yes/no]:  yes
Send peer a key in return[confirm]
Which one?
cisco2600-2?  [yes]:  yes
Public key for cisco2600-2:
 Serial Number B39FD8CE
 Fingerprint   21F9 665A 34A3 CA3F 92BD


router1(config)#crypto key exchange dss 10.2.0.20 cisco2600-1
Public key for cisco2600-1:
 Serial Number 593CA972
 Fingerprint   C9F6 7255 FF19 D7BD D8BD

Wait for peer to send a key[confirm]
Waiting ....

Public key for cisco2600-2:
 Serial Number B39FD8CE
 Fingerprint   21F9 665A 34A3 CA3F 92BD

Add this public key to the configuration?  [yes/no]:  yes
```

Both routers know now both keys. Like this they are able to encrypt and decrypt the traffic to and from the other router.

Instruction: *"Configure the routers to encrypt both TCP and UDP traffic between the two subnet 10.1.0.0/24 and 10.3.0.0/24. Make sure the routers use DES (Data Encryption Standard) algorithm with a Cipher Feedback Modus (CFB) of 64 bit."*

First a crypto access list has to be created. It is used to define which IP packets will be encrypted and which will not. The command `ip access-list extended <id>` creates and enters the access list configuration mode. `permit <protocol> <source> <destination>` allows to define which traffic we want to allow in our VPN. With `deny` we could also prevent some hosts from sending or receiving encrypted traffic. Because we do not have a rule for the 10.2.0.0 subnet all traffic from and to this subnet will not be encrypted.

```
router1(config)#ip access-list extended 100
router1(config-ext-nacl)#permit tcp 10.1.0.0 0.0.0.255 10.3.0.0 0.0.0.255
router1(config-ext-nacl)#permit udp 10.1.0.0 0.0.0.255 10.3.0.0 0.0.0.255
router1(config-ext-nacl)#exit
```

Next a crypto map has to be defined. Like with many commands before `crypto map <name> <seq number>` creates such a crypto map and enters the configuration mode. It combines the encryption algorithm (`set algorithm <algorithm>`) and the connection peer (`set peer <peer key>`) with the previously defined access list (`match address <access list id>`). According to the instruction we choose the DES algorithm with a 64 bit cypher feedback.

```
router1(config)#crypto map router1_map 1
% NOTE: This new crypto map will remain disabled until a peer
      and a valid access list have been configured.
router1(config-crypto-map)#set algorithm des cfb-64
router1(config-crypto-map)#match address 100
router1(config-crypto-map)#set peer cisco2600-2
router1(config-crypto-map)#exit
```

The last step in the VPN configuration is to apply the crypto map to an interface. Certainly the interface which is connected with the other router, alternatively the 10.2.0.0 subnet, has to be chosen.

```
router1(config)#interface FastEthernet0/0
router1(config-if)#crypto map router1_map
router1(config-if)#exit
```

To finish the entire procedure the steps to define the access list and crypto map and to apply the crypto map has to be repeated on router2 with the corresponding argument changes.

6.1.5 Testing the VPN

All the previous tests from the subsection 6.2.3 'Testing RIP' should be repeated and compared with the earlier results. After we made sure that all the connections are still working, we can start with the bandwidth measurement. The commands are exactly the same like before:

```
host3@host3:~$ NPtcp
Send and receive buffers are 16384 and 87380 bytes
(A bug in Linux doubles the requested buffer sizes)

host1@host1:~$ NPtcp -h host3 -o netpipe2.stats
Sending output to netpipe2.stats
Send and receive buffers are 16384 and 87380 bytes
(A bug in Linux doubles the requested buffer sizes)
Now starting the main loop
0:  1 bytes 73 times --> 0.01 Mbps in 1362.54 usec
1:  2 bytes 73 times --> 0.01 Mbps in 1362.99 usec
2:  3 bytes 73 times --> 0.02 Mbps in 1369.00 usec
3:  4 bytes 48 times --> 0.02 Mbps in 1373.42 usec
4:  6 bytes 54 times --> 0.03 Mbps in 1368.28 usec
...
72:  24579 bytes 3 times --> 1.61 Mbps in 116243.01 usec
73:  32765 bytes 3 times --> 1.66 Mbps in 150463.83 usec
74:  32768 bytes 3 times --> 1.65 Mbps in 151219.49 usec
75:  32771 bytes 3 times --> 1.65 Mbps in 151217.33 usec
```

With enabled encryption the routers now have to decrypt and encrypt a part of the traffic which is obviously not supported by special encryption hardware. Therefore the routers have to use their processors for these steps, significantly reducing the available bandwidth.

A second time we capture the login traffic between hosts1 and host3. The encryption tunnel does not prevent host2 from receiving the packets. But we understand the difference to the original setup by analysing the captured packet content and look for the sent password:

```
host2@host2:~$ tcpdump -r tcpdump2.log -XX
...
00:50:19.915449 IP 10.3.0.100.33184 > 10.1.0.100.telnet:  P 83:84(1) ack 110
win 5840 <nop,nop,timestamp 2102141193 1290346684>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 c4b8 4000 3f06 622f 0a03 0064 0a01 .5..@.?.b/...d..
   0x0020:  0064 81a0 0017 482c d275 5228 66c9 8018 .d....H,.uR(f...
   0x0030:  16d0 77d2 0000 0101 080a 7d4c 2109 4ce9 ..w.......}L!.L.
   0x0040:  20bc 28                                 ..(
00:50:19.979765 IP 10.3.0.100.33184 > 10.1.0.100.telnet:  P 84:85(1) ack
110 win 5840 <nop,nop,timestamp 2102141257 1290347497>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 c4b9 4000 3f06 622e 0a03 0064 0a01 .5..@.?.b....d..
   0x0020:  0064 81a0 0017 482c d276 5228 66c9 8018 .d....H,.vR(f...
   0x0030:  16d0 8164 0000 0101 080a 7d4c 2149 4ce9 ...d......}L!IL.
   0x0040:  23e9 8c                                 #..
00:50:20.236527 IP 10.3.0.100.33184 > 10.1.0.100.telnet:  P 85:86(1) ack 110
win 5840 <nop,nop,timestamp 2102141514 1290347521>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 c4ba 4000 3f06 622d 0a03 0064 0a01 .5..@.?.b-...d..
   0x0020:  0064 81a0 0017 482c d277 5228 66c9 8018 .d....H,.wR(f...
   0x0030:  16d0 714a 0000 0101 080a 7d4c 224a 4ce9 ..qJ......}L"JL.
   0x0040:  2401 0b                                 $..
00:50:20.427154 IP 10.3.0.100.33184 > 10.1.0.100.telnet:  P 86:87(1) ack 110
win 5840 <nop,nop,timestamp 2102141705 1290347778>
   0x0000:  00d0 5865 f560 00d0 5865 f540 0800 4510 ..Xe.`..Xe.@..E.
   0x0010:  0035 c4bb 4000 3f06 622c 0a03 0064 0a01 .5..@.?.b,...d..
   0x0020:  0064 81a0 0017 482c d278 5228 66c9 8018 .d....H,.xR(f...
   0x0030:  16d0 7489 0000 0101 080a 7d4c 2309 4ce9 ..t.......}L#.L.
   0x0040:  2502 71                                 %.q
```

This Cisco DSS encryption does not use an authentication or encryption technique from IPSec (like AH or ESP). Therefore the Ethernet, IP and TCP header are still the same like with the unencrypted connection. Only the payload of the packets of the defined transport protocols (in our case TCP and UDP) is encrypted. We used the same password 'reto' in this try again, but now, the values in the data field are encrypted and thus meaningless for us.

6.2 Conclusion

We have configured two Cisco routers to fit in our network, sniffed a password from plain traffic and compared the bandwidth of an encrypted and an unencrypted connection. The traffic rate is fallen from 7 Mbps to only around 1.7 Mbps because additional computation, namely encryption and decryption, has to be done by the router processors. On the other hand our transfered content could not be extracted anymore. Unfortunately even this encryption method does not give us a hundred percent security because the limited key length of 56 bit allows only a total number of 2^{56} possible keys. When we consider that a brute force attack would be able to check 10^{12} keys per second we would only need 20 hours to break the key. A modern Apple dual G5 2 GHz system already reaches the computation power of $3 * 10^6$ keys per second.

Chapter 7

Practical Experience

As teaching assistant in the Computer Networks lecture I was supervising the practical use of this VITELS module. During a couple of weeks about sixty students had to work with the newly migrated setup. Both installations, the one with the Cisco 2600 routers and the TFTP bootstrapping and the other with the Cisco 3600 routers and the original bootstrapping, run actually quite smooth. We did not find any problems concerning the new setup. Nonetheless there are a few weaknesses in the design of the whole script layer.

7.1 Issues in practical use

- The delayed synchronisation between the logged-in users and the password generation, invoked by cron, is not very suitable (see subsection 4.3.6). It can lead to temporary 'Authentication failed' responses to a recently logged-in module user.
- The reset of the router is not instantly reported to the module user. Because the current Web portal cannot process incoming messages, the current status of the reset activity cannot be communicated to the user. The upcoming new Web portal will hopefully take care of this.
- The used Mindterm Java applet for connecting to the device terminals does not run very smoothly on all platforms. This was the point where the biggest number of users reported some problems. When the SSH connection unexpectedly aborts, the applet stays in a state where it is not possible anymore to properly use or quit it. Only a restart of the Web browser can make Mindterm run again. Also it does not interpret magic keys (e.g. to quit Minicom) sent by the keyboard. A solution could be to officially provide the temporary password by the Web portal. Already now it is possible to get the password from the Web site's source code, but this is not very convenient for most of the users. The password would allow the users to log in via an ordinary terminal application and take advantage of their more comfortable handling. Further a transfer of configuration and log files via `scp` would be possible too. At the moment this is still done by simply copy and paste the file content to and from the Mindterm console.

7.2 Troubleshooting Problems

The scripts and application described in section 4.4 'Testing the Setup' help to notify the module administrator when problems occur. To track down system errors and bugs it is further useful to check up the log files. For general issues with the Linux system the ordinary log files in */var/log/* should be consulted. The most important files on a Debian system are */var/log/messages* for kernel related feedback, */var/log/syslog* for userspace application messages and */var/log/auth.log* for authentication failures.

The terminal emulation Minicom writes its log file to */root/minicom.log*. It should be checked during problems concerning the connection between the portal server and the routers.

Also the script for the user authentication and session handling (see section 4.3.6) is able to output some logging and debugging information. It creates a log file */root/update_shadow.log*. The script defines a variable $debug that should be changed to "0" to enable additional task by task output:

```
my $debug = "0"
```

Listing 7.1: /root/update_shadow_ldap.cnds.unibe.ch_ssl.pl (section of interest)

With all this information it should easily be possible to find any system issues and detect its originators.

References

[1] Thomas Spreng: IP Security Module for VITELS, Informatikprojekt RVS, 2003

[2] SWITCH PKI: http://www.switch.ch/pki/

[3] NetPIPE: http://www.scl.ameslab.gov/netpipe/

[4] TCPDUMP: http://www.tcpdump.org/

[5] Xen: http://www.cl.cam.ac.uk/Research/SRG/netos/xen/

[6] User-Mode Linux: http://user-mode-linux.sourceforge.net/

[7] Conrad Electronics: http://www.conrad.ch/

[8] Extended 3 Filesystem: http://www.redhat.com/support/wpapers/redhat/ext3/

[9] XFS: http://oss.sgi.com/projects/xfs/

[10] GRUB: http://www.gnu.org/software/grub/

[11] MindTerm: http://www.appgate.com/products/80_MindTerm/

[12] Minicom: http://alioth.debian.org/projects/minicom/

[13] Inetd: http://www.linuxfibel.de/inetd.htm

[14] sudo: http://www.courtesan.com/sudo/

[15] Stunnel: http://www.stunnel.org/

[16] Exim: http://www.exim.org/

[17] ssmtp: http://packages.qa.debian.org/s/ssmtp.html

[18] Sendmail: http://www.sendmail.org/

[19] Logwatch: http://www2.logwatch.org:8080/

[20] XMODEM: http://www.techfest.com/hardware/modem/xymodem.htm

[21] tftpd-hpa: http://packages.qa.debian.org/t/tftp-hpa.html

Appendix A

Bibliography

- David Jud, Drei Module für angewandtes Lernen von Computernetzwerken, 2001/2003 (http://www.iam.unibe.ch/%7Ervs/research/pub_files/Ju03.pdf)
- Weyland Attila, Vitels Hands-On Session Setup, 2004 (http://www.vitels.ch/doc/vitels_hands_on_setup.pdf)
- Zimmerli Stefan, Internetportal für Computernetze-Praktika, Diplomarbeit 2002 (http://www.iam.unibe.ch/˜szimmer/diplomarbeit_stefan_zimmerli.pdf)
- Debian GNU/Linux Installation Guide (http://www.debian.org/releases/stable/i386/)
- Software Configuration Guide for Cisco 2600 Series, Cisco 3600 Series and Cisco 3700 Series Router (http://www.cisco.com/application/pdf/en/us/guest/products/ps259/c2001/ccmigration_09186a00801f6f6b.pdf)
- IP Routing (http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/np1_c/1cprt1/1crip.pdf)
- IPSec Network Security (http://www.cisco.com/univercd/cc/td/doc/product/software/ios113ed/113t/113t_3/ipsec.pdf)
- Configuring Network Data Encryption with Router Authentication (http://www.cisco.com/univercd/cc/td/doc/product/software/ios112/112cg_cr/2cbook/2cencryp.pdf)
- Cisco Encryption Technology Commands (http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/secur_r/srprt4/srencryp.pdf)
- Cisco 2600 and 3600 Hints and Tricks (ID 12817, http://www.cisco.com/warp/public/701/59.pdf)
- Virtual Configuration Register (http://www.cisco.com/univercd/cc/td/doc/product/access/acs_fix/cis3000/c3k2him/20586.pdf)

- Booting Commands
  (http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/fun_r/frprt2/frreboot.pdf)

- ROMmon Recovery for the Cisco 3600 and 3700 Series Router
  (ID 15080, http://www.cisco.com/warp/public/130/recovery_c3600.pdf)

- How to Choose a Cisco IOS Software Release
  (ID 15071, http://www.cisco.com/warp/public/130/choosing_ios.pdf)

- Loading Cisco IOS Software
  (http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/relnote/fprn/loadswfp.pdf)

- TFTP Server Selection & Use
  (ID 48700, http://www.cisco.com/warp/public/63/tftp-server.pdf)

- How to Download a Software Image to a Cisco 2600 through TFTP Using tftpdnld ROMmon Command
  (ID 12714, http://www.cisco.com/warp/public/471/76.pdf)

- Xmodem Console Download Procedure Using ROMmon
  (ID 15085, http://www.cisco.com/warp/public/130/xmodem_generic.pdf)

Appendix B

Linux Configuration Files

B.1 */etc/fstab*

```
# /etc/fstab: static file system information.
#
# <file system> <mount point>    <type>   <options>          <dump><pass>
/dev/hda1       /boot            ext3     noauto             0     2
/dev/hda2       none             swap     sw                 0     0
/dev/hda3       /                xfs      defaults           0     1
/dev/hda5       /var             xfs      defaults           0     2
/dev/hda6       /backup          xfs      noauto             0     2

/dev/hdb        /media/cdrom0    iso9660  ro,user,noauto     0     0

proc            /proc            proc     defaults           0     0
```

Listing B.1: /etc/fstab

B.2 */etc/hosts*

```
127.0.0.1       localhost.localdomain   localhost

10.1.0.100      host1                   host1

10.1.0.10       router1_0               router1_0
10.2.0.10       router1_1               router1_1

10.2.0.100      host2                   host2

10.2.0.20       router2_0               router2_0
10.3.0.20       router2_1               router2_1

10.3.0.100      host3                   host3

10.1.0.1        gridlab06.unibe.ch      gridlab06
```

Listing B.2: /etc/hosts

Appendix C

Portal Server Custom Scripts

C.1 Overview

C.2 */bin/host1*

```
#!/bin/bash
# is called when user host1 logs into gridlab06.unibe.ch
# user is redirected to host1@host1 (needs authorized ssh keypair)

ssh host1
exit
```

Listing C.1: /bin/host1

C.3 */bin/router1*

```
#!/bin/bash
# filename: /bin/router1
# real tty connected to router1

# first: kill the existing jobs
sudo /root/namekill $USER minicom

# second: wait until minicom has shut down
sleep 6s

# third: start new minicom connection
/usr/bin/minicom -C /home/router1/minicom.log ttyS0
exit
```

Listing C.2: /bin/router1

C.4 */root/namekill*

```
#!/usr/bin/perl

use strict;

&usage unless (@ARGV == 2);

my $signal = 'TERM';
if (@ARGV > 2) {
    $signal = shift @ARGV;
}

my $user = shift @ARGV;
my $name = shift @ARGV;

my @tasks = `ps -u $user`;
foreach (@tasks) {
    if (/^\s*(\S+).*$name/) {
        kill $signal, $1;
    }
}


sub usage {
    print "Usage: $0 [ signal ] user taskname\n\n";
    exit 1;
}
```

Listing C.3: /root/namekill

C.5 */root/update_shadow_ldap.cnds.unibe.ch_ssl.pl*

```perl
#!/usr/bin/perl

# This perl script should be run as a cron job. It reads a
# module’s current user and - if the user changed - it updates
# the shadow file, kills the old session and resets the hosts
# and routers.
#
# author:   Thomas Jampen <jampen@iam.unibe.ch>
# created:  20020203
# modified: 20020501
# modified: 20030515 stefan zimmerli <szimmer@iam.unibe.ch>
#           changed ldap settings to work over stunnel
# modified: 20030621 stefan zimmerli <szimmer@iam.unibe.ch>
#           commented getting password from ldap
#           included functions from PwGenLib to generate a onetime-password
#           and to check user changes by saving the currentuser into
#           /etc/current_user
# modified: 20050808 reto gantenbein <gantenbe@iam.unibe.ch>
#           removed needless users cisco2600 and cisco3600 from %users
# modified: 20051129 reto gantenbein <gantenbe@iam.unibe.ch>
#           added lines for resetting module (module_reset.sh)

use strict;
use Net::LDAP qw(:all);
require PwGenLib;

# variable definitions
# ====================

my $host    = "localhost";
my $port    = "636";
my $basedn  = "ou=Modules,o=VITELS,c=CH";
my $binddn  = "uid=labportal,ou=Staff,o=VITELS,c=CH";
my $bindpw  = "l3s3r3cht";

my $dummydn = "uid=dummy,ou=users,o=Universitaet Bern,c=CH";
# ipsec inf_vitels
my $mid     = "6";
my %users = ("host1", "host1", "host2", "host2", "host3", "host3",
            "router1", "router1", "router2", "router2");
my $debug= "1";
#my $debug= "0";


# connect to the LDAP server
# ==========================
print "Connecting to the server\n" if $debug;
my $ldap = Net::LDAP->new("$host:$port") or die "$@";


# bind
# ====
my $bind = $ldap->bind($binddn, password => "$bindpw", version => 3);
unless ($bind)
{
    die "Unable to bind to server!\n";
}
```

```


# check module’s current user
# ==========================
my $search = $ldap->search(base   => "mid=$mid,$basedn",
                           scope  => "sub",
                           deref  => "find",
                           filter => "(uid=*)",
                           attrs  => ("uid", "userpassword"));

my @users = $search->entries;
my $user;

foreach $user (@users)
{
    my $user_attr;
    my $current_user;
    my $current_pwd;
    my $pwd;
    my $pwd_len;
    my @line;

    # search for the attributes ’uid’, ’userpassword’
    # we need them later
    foreach $user_attr ($user->attributes)
    {
    if ($user_attr eq "uid")
    {
        $current_user = $user->get_value($user_attr);
        print "current_user= $current_user\n" if $debug;
    }
    }

    # get the saved current user
    open(USER_OLD,"</etc/current_user")
    || die "Unable to read current_user file!";
    my $last_user= "";
    my $line= <USER_OLD>;
    close (USER_OLD);
    my $len= length($line);
    print "line=$line,len=$len\n" if $debug;
    $last_user= substr($line,0,$len-1);

    print "last_user= $last_user\n" if $debug;

    # still the same user is current user -> do nothing
    #
    if ($last_user eq $current_user)
    {
    print ":$last_user: EQ :$current_user:\n" if $debug;
    }

    # the current user has changed -> save uid and kill processes
    #
    else
    {
    print ":$last_user: NOT EQ :$current_user:\n" if $debug;

    open(USER_NEW,">/etc/current_user")
        || die "Unable to read current_user file!";
    print USER_NEW $current_user,"\n";
    close (USER_NEW);
```

```perl

    print "get new password\n" if $debug;
    # get a onetime password
    my $plainpw= &PwGenLib::generate_password;
    print "plain:",$plainpw,"\n" if $debug;

    # make a md5hash out of the password
    my $md5pw= &PwGenLib::make_md5_password($plainpw);
    print "md5pw:",$md5pw,"\n" if $debug;

    # save the plain password (we will read it from the php portal frontend)
    &PwGenLib::write_plain_password($plainpw);
        # save the md5-password into /etc/shadow
    &PwGenLib::write_shadow_password($md5pw);
    print "new passwords written\n" if $debug;

    # kill processes of the previous user
    foreach (keys(%users))
    {
        print "killing $_\n" if $debug;
        `/root/namekill $_ $users{$_}`;
    }

        print "reset hosts" if $debug;
    # reset hosts
    `/root/module_reset.sh`;

    print "backup shadow file\n" if $debug;
    # backup old shadow file and replace with new version

    my $backup_shadow_command=
        "cp ".$PwGenLib::SHADOW_FILE." ".$PwGenLib::SHADOW_FILE.".OLD";

    my $move_new_shadow_command=
        "mv ".$PwGenLib::SHADOW_FILE.".NEW ".$PwGenLib::SHADOW_FILE;

    my $set_permissions_command=
        "chgrp shadow ".$PwGenLib::SHADOW_FILE." && "
                ."chmod 640 ".$PwGenLib::SHADOW_FILE." && "
                ."chmod 600 ".$PwGenLib::SHADOW_FILE.".OLD";

    `$backup_shadow_command`;
    `$move_new_shadow_command`;
    `$set_permissions_command`;
    }
}


# disconnect
# ==========
$ldap->unbind;

print "disconnect from server\n" if $debug;

exit 0;
```

Listing C.4: /root/update_shadow_ldap.cnds.unibe.ch_ssl.pl

C.6 */usr/local/bin/stunnel_ldap.sh*

```
#!/bin/bash
# redirects and encrypts LDAP connection on localhost to LDAP server

/usr/sbin/stunnel -c -d 636 -r ldap.cnds.unibe.ch:636
```

Listing C.5: /usr/local/bin/stunnel_ldap.sh

C.7 */root/module_reset.sh*

```
#!/bin/bash
# script to reset the hosts and routers
# 29.11.2005 by reto gantenbein <gantenbe@iam.unibe.ch>

# empty the home directories of the hosts
for USER in "host1" "host2" "host3"
do
  sudo -u $USER ssh $USER@$USER rm -rf /home/$USER/*
  sudo -u $USER ssh $USER@$USER rm /home/$USER/.bash_history
done

# reset routers
for ROUTER in "router1" "router2"
do
  /var/php_safe_mode_bin/pw_reset.pl $ROUTER
done

exit 0
```

Listing C.6: /root/module_reset.sh

C.8 */root/module_test.sh*

```
#!/bin/bash

# Script to test basic 'IP Security' module services
# Reto Gantenbein (gantenbe@iam.unibe.ch)

LOGFILE=/root/module_status.log
ALERT_MAIL=gantenbe@iam.unibe.ch
CC_MAIL=bernoull@iam.unibe.ch

#
# Test Hosts
#
for HOST in host1 host2 host3 ; do
  ping -c1 $HOST >> /dev/null 2>&1
  if [ $? -eq 1 ] ; then
    echo "$HOST does not respond!" >> $LOGFILE
  fi
done

#
# Test Apache2
#
APACHE_PID=$(pidof apache2)
APACHE_PORT=$(netstat -napt | grep 0.0.0.0:443)

if [ -z "$APACHE_PID" -o -z "$APACHE_PORT" ] ; then
  echo "Apache2 is not running properly!" >> $LOGFILE
fi

#
# Test SSH
#
SSH_PID=$(pidof sshd)
SSH_PORT=$(netstat -napt | grep 0.0.0.0:22)

if [ -z "$SSH_PID" -o -z "$SSH_PORT" ] ; then
  echo "SSH is not running properly!" >> $LOGFILE
fi

#
# Send alert mail
#
if [ -f $LOGFILE ] ; then
   mail -s "$HOSTNAME.unibe.ch: IP-Security Module Alert" -c $CC_MAIL $ALERT_MAIL \
   < $LOGFILE
   rm $LOGFILE
fi
```

Listing C.7: /root/module_test.sh

C.9 */root/PwGenLib.pm*

```perl
##############################################################################
#
# gateway for a remote laboratory
#
# file:    PwGenLib.pm
#
# purpose:
#
#
# depends:
#
# author:  stefanzimmerli.com, software@stefanzimmerli.com
#
# version:
#   01  01-03-2003 steffu initial version (quick hack)
#   02  10-03-2003 steffu implemented generate_password
#   03  30-03-2003 steffu implemented make_md5_password,write_plain_password
#   04  05-04-2003 steffu implemented write_shadow_password
#   05  21-06-2003 steffu did a lot of testing. found some bugs in the
#                         generate password function
#   06  23-06-2003 steffu fixed the bugs. everything ist working!
#
##############################################################################

package PwGenLib;
use strict;

$PwGenLib::VERSION="PwGenLib V0.6";
$PwGenLib::PWLENGTH=8;
@PwGenLib::CHARS=('A'..'Z','a'..'z','0'..'9');
$PwGenLib::CURRENT_PLAIN_PW_FILE="/etc/current_pw";
$PwGenLib::CURRENT_USER_FILE="/etc/current_user";
$PwGenLib::SHADOW_FILE="/etc/shadow";
@PwGenLib::USERS=('host1','host2','host3','cisco2600',
                  'cisco3600','router1','router2');


##############################################################################
sub write_shadow_password
{
    my $_password= shift;
    #print "md5pw:",$_password,"\n";
    #print "users",@PwGenLib::USERS,"\n";

    open(SHADOW_OLD,$PwGenLib::SHADOW_FILE) or
        die "cannot open file $PwGenLib::SHADOW_FILE!";

    open(SHADOW_NEW,">".$PwGenLib::SHADOW_FILE.".NEW")  or
        die "cannot open file $PwGenLib::SHADOW_FILE.NEW";

    while (my $_line=<SHADOW_OLD>)
    {
        #print "ORIG$_line";
        # search for uid and pw
        #         uid  : pw  :number:zero:number:number:::
        $_line=~/([\w-]+)\:(.+?)(\:\d+\:0\:\d+:\d+\:\:\:)$/;
        my $_user= $1;
        my $_pw  = $2;
        my $_rest= $3;

        #print "NEW $_user $_pw$_rest\n";

        if (grep(/$_user/,@PwGenLib::USERS) > 0)
            {
                #print "labuser found:",$_user,",pw:",$_pw,"\n";
                print SHADOW_NEW "$_user:$_password$_rest","\n";
            }
            else
            {
                #print "system user found:",$_user,",pw:",$_pw,"\n";
                print SHADOW_NEW "$_user:$_pw$_rest","\n";
            }
    }

    close (SHADOW_NEW);
    close (SHADOW_OLD);
}

##############################################################################
sub write_plain_password
{
    my $_password= shift;

    open(PLAINPW, ">".$PwGenLib::CURRENT_PLAIN_PW_FILE) or
        die "cannot open file $PwGenLib::CURRENT_PLAIN_PW_FILE!";

    print PLAINPW $_password;

    close(PLAINPW);
}

##############################################################################
sub make_md5_password
{
    my $_plain_password= shift;
    my $_salt= "\$1\$";
    for (my $i=0;$i<4;$i++)
    {
        $_salt.= $PwGenLib::CHARS[rand(scalar(@PwGenLib::CHARS)-1)];

    }
    #print "salt:",$_salt, "\n";
    #print "plain:",$_plain_password, "\n";

    my $_md5_password= crypt($_plain_password,$_salt);

    return $_md5_password;
}

##############################################################################
sub get_version
{
    return $PwGenLib::VERSION;
}

##############################################################################
sub generate_password
{

    my $_password= '';

    #print scalar(@PwGenLib::CHARS),"\n";
    for (my $i=0;$i<$PwGenLib::PWLENGTH;$i++)
    {
        $_password.= $PwGenLib::CHARS[rand(scalar(@PwGenLib::CHARS)-1)];

    }
    return $_password;
}

##############################################################################
1;
```

Listing C.8: /root/PwGenLib.pm

C.10 */usr/share/perl5/RelaisLib.pm*

```
################################################################################
#
# gateway for a remote laboratory
#
#
# file:    RelaisLib.pm
#
# purpose: simple perl api to control the serial line 8-port relais card from
#          conrad. with this api we will powercycle our two cisco routers to
#          delete the set passwords on our cisco routers
#
# depends: on the module Device:SerialPort
#
# author:  stefan.zimmerli@iam.unibe.ch
#
# version:
#   01  03-07-2002 steffu initial version (quick hack)
#   02  04-07-2002 steffu wrote first subroutines
#   03  06-07-2002 steffu made tests, added more subs
#   04  06-07-2002 steffu finished and tested the card api
#   05  06-07-2002 steffu wrote subs to powercycle the routers.
#   06  08-09-2002 steffu made two files relais_lib_XY.pl and relais_test_01.pl
#                         problem: reading the card status gave not what
#                         i set before...
#   07  18-09-2002 steffu made package RelaisLib in file RelaisLib.pm
#   08  20-09-2002 steffu did a lot of testing to solve those package scope
#                         problems now reading the card status gives what
#                         i set before
#
################################################################################

package RelaisLib;

use strict;
use Device::SerialPort qw( :PARAM :STAT 0.07);

$RelaisLib::RELAIS_CARD_SERIAL_DEVICE= "/dev/ttyS2";
$RelaisLib::NOP      = 0;
$RelaisLib::INIT     = 1;
$RelaisLib::GET      = 2;
$RelaisLib::SET      = 3;
$RelaisLib::ADDR     = 1;
$RelaisLib::POWER_OFF= 0;
$RelaisLib::POWER_ON = 1;

# debugging/logging of the library functions to standard output
# 0: logging, 1: no logging
# set them here OR in the perl program that uses this library!
#
#$RelaisLib::NODEBUG= 0;
$RelaisLib::NODEBUG= 1;

################################################################################
sub get_relais_status
{
    # input:  tty got from relaiscard_tty_open
    # output: relais status in decimal (0: all relais off, 1: relais 0 on,
    #         2: relais 1 on, 255: all relais on)

    my $mytty = shift;

    #print "\n==== INIT CARD ====\n";
    RelaisLib::relaiscard_send_command($mytty, $RelaisLib::ADDR,
    $RelaisLib::INIT,0);
    sleep (1);

    #print "\n==== GET PORT STATUS ====\n";
    my @portstatus= RelaisLib::relaiscard_send_command($mytty,
    $RelaisLib::ADDR, $RelaisLib::GET,0);

    #print "port status=$portstatus[2]\n";
    $portstatus[2];
}

################################################################################
sub router_power
{
    # input:  tty, router_number($C2600,$C3600), power($POWER_OFF,$POWER_ON)
    # output: 1 if ok, 0 otherwise
    print "---- sub router_power starts ----\n" unless $RelaisLib::NODEBUG;

    my ($mytty,$myrouter_number,$mypower_status)= @_;
    my @answer= relaiscard_send_command($mytty, $RelaisLib::ADDR,
    $RelaisLib::GET,0);

    #print "port status of the relais card: $answer[2]\n";
    print "---- sub router_power ends ----\n" unless $RelaisLib::NODEBUG;
}

################################################################################
sub relaiscard_send_command
{
    # input:  tty,card_address,command,parameter
    # output: decoded answer frame from relaiscard

    print "---- sub relaiscard_send_command starts ----\n"
    unless $RelaisLib::NODEBUG;
    my ($mytty,$card_address,$command,$parameter)= @_;

    my @answer= 0;

    my $data= encode_frame($command,$card_address,$parameter);
    my $count_out= $mytty->write($data);
    warn "write failed\n" unless ($count_out);
    warn "write incomplete\n" if ($count_out != length($data));

    sleep(1);
    my ($count_in,$input)= $mytty->read(4);
    if (check_frame($input))
    {
        @answer= decode_frame($input);
        print "frame bytes \tsent\treceived\n" unless $RelaisLib::NODEBUG;
        print "command \t$command\t$answer[0]\n" unless $RelaisLib::NODEBUG;
        print "cardaddr\t$card_address\t$answer[1]\n"
        unless $RelaisLib::NODEBUG;
        print "data\t\t$parameter\t$answer[2]\n" unless $RelaisLib::NODEBUG;
    }

    if ($command==$RelaisLib::INIT)
```

```perl
    {
        my ($count_in_init,$input_init)= $mytty->read(4);
        if (check_frame($input_init))
        {
            my @second_init_frame= decode_frame($input_init);
            #print "second init frame @second_init_frame\n";
        }
    }

    print "---- sub relaiscard_send_command ends ----\n"
    unless $RelaisLib::NODEBUG;

    @answer;
}

##############################################################################
sub relaiscard_tty_close
{
    # input:  the port handle got from relaiscard_tty_open
    # output: void

    my $myport= shift(@_);
    $myport->close || die "failed to close $!\n";
    undef $myport;
}

##############################################################################
sub relaiscard_tty_open
{
    # input:  the serial device where the relais card is attached to
    #         (/dev/ttyS2)
    # output: the port handle

    my $device= shift;
    my $myport= new  Device::SerialPort($device)
    || die "cant open $device $!\n";
    $myport->user_msg("ON");
    $myport->databits(8);
    $myport->baudrate(19200);
    $myport->parity("none");
    $myport->stopbits(1);
    $myport->write_settings || undef $myport;

    $myport;
}

##############################################################################
sub encode_frame
{
    # input:  three integers
    # output: four-byte-string with checksum

    print "---- sub encode_frame starts ----\n" unless $RelaisLib::NODEBUG;

    my($byte0,$byte1,$byte2)= @_;
    my $checksum = $byte0 ^ $byte1 ^$byte2;

    my $frame= pack("CCCC",$byte0,$byte1,$byte2,$checksum);

    print "encoded data=($byte0/$byte1/$byte2/$checksum),
    packed data=<$frame>\n" unless $RelaisLib::NODEBUG;

    print "---- sub encode_frame ends ----\n" unless $RelaisLib::NODEBUG;

    $frame;
}

################################################################################
sub decode_frame
{
    # input:  four-byte-string with checksum
    # output: array with three integers (command, address, data)

    my($frame)= shift (@_);
    my($frame_length)= length($frame);

    #print "---- sub decode_frame starts ----\n" unless $RelaisLib::NODEBUG;
    #print "frame=$frame, length=$frame_length\n" unless $RelaisLib::NODEBUG;
    my @data= unpack("CCCC",$frame);

    #print "@data\n" unless $RelaisLib::NODEBUG;
    #print "---- sub decode_frame ends ----\n" unless $RelaisLib::NODEBUG;
    @data;
}

################################################################################
sub check_frame
{
    # input:  four-byte-string with checksum
    # output: 1 if checksum is ok
    #         0 otherwise
    my($in_frame)= shift (@_);
    my($frame_length)= length($in_frame);

    my @frame= unpack("CCCC",$in_frame);

    if (($frame[0] ^ $frame[1] ^ $frame[2]) == $frame[3])
    {
        #print "frame ok\n" unless $RelaisLib::NODEBUG;
        return 1;
    }
    else
    {
        #print "frame NOT ok\n" unless $RelaisLib::NODEBUG;
        return 0;
    }
}

################################################################################

1;
```

Listing C.9: /usr/share/perl5/RelaisLib.pm

C.11 */usr/share/perl5/RouterLib.pm*

```
############################################################################
#
# gateway for a remote laboratory
#
# file: RouterLib.pm
#
# purpose: provides helper functions to send commands to the routers
# over the serial line
#
# depends: on the module Device:SerialPort
#
# author: stefan.zimmerli@iam.unibe.ch
#
# history:
#    01  15-10-2002 Stefan Zimmerli
#    02  11-08-2005 Reto Gantenbein <gantenbe@iam.unibe.ch>
#                   -> added bootstrapping from tftp (cisco2600)
#
############################################################################

package RouterLib;
require RelaisLib;

use Device::SerialPort qw( :PARAM :STAT 0.07);

$RouterLib::LOCKFILE_PATH= '/tmp/';
$RouterLib::MAX_TIME= 360;
$RouterLib::ROUTER1_DEVICE= "/dev/ttyS0";
$RouterLib::ROUTER2_DEVICE= "/dev/ttyS1";
$RouterLib::ROUTER1_OFF = 254;
$RouterLib::ROUTER1_ON  =   1;
$RouterLib::ROUTER2_OFF = 253;
$RouterLib::ROUTER2_ON  =   2;
$RouterLib::ROUTER1     = "router1";
$RouterLib::ROUTER2     = "router2";
$RouterLib::ROUTER1_TYP = "cisco3600";
$RouterLib::ROUTER2_TYP = "cisco3600";

sub create_lockfile
{
    my $mylockfile= $RouterLib::LOCKFILE_PATH;
    $mylockfile.= shift;
    print "create: lockfile=$mylockfile\n" unless $RouterLib::NODEBUG;
    open (LOCKFILE,">".$mylockfile)
    or die "can not make lockfile $mylockfile: $!\n";
    close (LOCKFILE);
}

sub delete_lockfile
{
    my $mylockfile= $RouterLib::LOCKFILE_PATH;
    $mylockfile.= shift;
    print "delete: lockfile=$mylockfile\n" unless $RouterLib::NODEBUG;
    unlink($mylockfile) or die "can not unlink lockfile $mylockfile: $!\n";

}

```

```perl
sub exist_lockfile
{
    my $mylockfile= $RouterLib::LOCKFILE_PATH;
    $mylockfile.= shift;
    print "exist: lockfile=$mylockfile\n" unless $RouterLib::NODEBUG;
    return (-e $mylockfile);
}

sub is_lockfile_new
{
    my $mylockfile= $RouterLib::LOCKFILE_PATH;
    $mylockfile.= shift;
    print "is new: lockfile=$mylockfile\n"  unless $RouterLib::NODEBUG;
    my @mylockfile_stats= stat($mylockfile);
    if ((time-$mylockfile_stats[9]) < $RouterLib::MAX_TIME) {
        return 1;
    } else {
        return 0;
    }
}

sub send_router_command
{
    # input: port (got from "new  Device::SerialPort($cisco2600_device)"),
    #        command string
    #        time to sleep in seconds after sending command
    # output: void

    my $_port= shift;
    my $_command= shift;
    my $_time= shift;
    print "command=$_command\n sending it and wait $_time seconds..."
    unless $RouterLib::NODEBUG;
    my $_count_out= $_port->write($_command);
    warn "write failed\n" unless ($_count_out);
    warn "write incomplete\n" if ($_count_out != length($_command));
    sleep($_time);
    print "done\n\n"  unless $RouterLib::NODEBUG;
}

sub reset_router
{
    my $myrouter= shift;
    print "reset router $myrouter\n"  unless $RouterLib::NODEBUG;

    my $tty= RelaisLib::relaiscard_tty_open(
    $RelaisLib::RELAIS_CARD_SERIAL_DEVICE);

    my $status= RelaisLib::get_relais_status($tty);
    my $myport= '';

    # powercycle device of router ONE
    if ($myrouter eq $RouterLib::ROUTER1) {

    my $new_status= $status &  $RouterLib::ROUTER1_OFF;
        RelaisLib::relaiscard_send_command($tty, $RelaisLib::ADDR,
        $RelaisLib::SET, $new_status);
    sleep(2);

    my $power_on = $new_status | $RouterLib::ROUTER1_ON;

        RelaisLib::relaiscard_send_command($tty, $RelaisLib::ADDR,
        $RelaisLib::SET, $power_on);

        RelaisLib::relaiscard_tty_close($tty);
    sleep(13);

    $myport= new  Device::SerialPort($RouterLib::ROUTER1_DEVICE)
        || die "cant open $RouterLib::ROUTER1_DEVICE $!\n";

    $reset_type=$RouterLib::ROUTER1_TYP;
    }

    # powercycle device of router TWO
    if ($myrouter eq $RouterLib::ROUTER2) {

        my $new_status= $status & $RouterLib::ROUTER2_OFF;

        RelaisLib::relaiscard_send_command($tty, $RelaisLib::ADDR,
        $RelaisLib::SET, $new_status);
        sleep(2);

        my $power_on = $new_status | $RouterLib::ROUTER2_ON;

        RelaisLib::relaiscard_send_command($tty, $RelaisLib::ADDR,
        $RelaisLib::SET, $power_on);

        RelaisLib::relaiscard_tty_close($tty);

        sleep(16);

        $myport= new  Device::SerialPort($RouterLib::ROUTER2_DEVICE)
        || die "cant open $RouterLib::ROUTER2_DEVICE $!\n";

        $reset_type=$RouterLib::ROUTER2_TYP;
    }

    if ($reset_type eq "cisco2600") {

        # scripts for cisco2600
        $myport->user_msg("ON");
        $myport->databits(8);
        $myport->baudrate(9600);
        $myport->parity("none");
        $myport->stopbits(1);
        $myport->write_settings || undef $myport;

        $myport->pulse_break_on(1000);

        send_router_command($myport,"confreg 0x2140\n",1);
        send_router_command($myport,"reset\n",10);

        if ($myrouter eq $RouterLib::ROUTER1) {
            send_router_command($myport,"IP_ADDRESS=10.2.0.10\n",1);
        } else {
            send_router_command($myport,"IP_ADDRESS=10.2.0.20\n",1);
        }

        send_router_command($myport,"IP_SUBNET_MASK=255.255.255.0\n",1);
        send_router_command($myport,"DEFAULT_GATEWAY=10.2.0.1\n",1);
        send_router_command($myport,"TFTP_SERVER=10.2.0.1\n",1);
        send_router_command($myport,"TFTP_FILE=currentIOS\n",1);
        send_router_command($myport,"tftpdnld\n",2);
        send_router_command($myport,"yes\n",170);
        send_router_command($myport,"boot flash:currentIOS\n",80);
        send_router_command($myport,"no\n",1);
    }

    if ($reset_type eq "cisco3600") {

        # scripts for cisco3600
        $myport->user_msg("ON");
        $myport->databits(8);
        $myport->baudrate(9600);
        $myport->parity("none");
        $myport->stopbits(1);
        $myport->write_settings || undef $myport;

        sleep(5);
        $myport->pulse_break_on(1000);

        # resetting the router old fashion way
        send_router_command($myport,"confreg 0x2142\n",1);
        send_router_command($myport,"reset\n",100);
        send_router_command($myport,"no\n",2);
        send_router_command($myport,"\r",2);
        send_router_command($myport,"\r",2);
        send_router_command($myport,"enable\n",1);
        send_router_command($myport,"configure\n",1);
        send_router_command($myport,"terminal\n",1);
        send_router_command($myport,"config-register 0x2102\n",1);
        send_router_command($myport,"exit\n",1);
        send_router_command($myport,"copy running-config startup-config\n",1);
        send_router_command($myport,"startup-config\n",1);
        send_router_command($myport,"reload\n",1);
        send_router_command($myport,"\n",120);
        send_router_command($myport,"\r",1);
    }

    $myport->close || die "failed to close $device";
    undef $myport;
}

1;
```

Listing C.10: /usr/share/perl5/RouterLib.pm

C.12 */var/php_safe_mode_bin/pw_reset.pl*

```
#!/usr/bin/perl

use Sys::Syslog;
require RelaisLib;
require RouterLib;

$RelaisLib::NODEBUG= 1;
$RouterLib::NODEBUG= 1;

openlog('pw_reset ','pid','syslog');

my $router= shift;
my $RouterResetCmd= '';
my $lockfile= '';

syslog('debug',"pw_reset.pl start");
syslog('debug',"input=$router\n");

if (($router eq $RouterLib::ROUTER1) || ($router eq $RouterLib::ROUTER2)) {

    syslog('debug',"router=$router\n");

    $lockfile= $router;

    if (RouterLib::exist_lockfile($lockfile))
    {
        syslog('debug',"lockfile $lockfile exists\n");

        if (RouterLib::is_lockfile_new($lockfile))
        {
            syslog('debug',"lockfile $lockfile is new. reset in progress!\n");
        }
        else
        {
            syslog('debug',"2 lockfile $lockfile old deleting it\n");
            RouterLib::delete_lockfile($lockfile);

            syslog('debug',"2 creating lockfile $lockfile\n");
            RouterLib::create_lockfile($lockfile);

            syslog('debug',"2 performing reset\n");
            RouterLib::reset_router($router);
            syslog('debug',"2 reset done\n");

            syslog('debug',"2 deleting lockfile $lockfile\n");
            RouterLib::delete_lockfile($lockfile);
        }

    }
    else
    {
        syslog('debug',"1 creating lockfile $lockfile\n");
        RouterLib::create_lockfile($lockfile);

        syslog('debug',"1 performing reset\n");
        RouterLib::reset_router($router);
        syslog('debug',"1 reset done\n");

        syslog('debug',"1 deleting lockfile $lockfile\n");
        RouterLib::delete_lockfile($lockfile);
    }
}

syslog('debug',"pw_reset done");
closelog;
```

Listing C.11: /var/php_safe_mode_bin/pw_reset.pl