FACHHOCHSCHULE REGENSBURG
UNIVERSITY OF APPLIED SCIENCES
HOCHSCHULE FÜR
TECHNIK
WIRTSCHAFT
SOZIALWESEN

Diplomarbeit

Mögliche Szenarien für das Virtuelle Unix Labor

Fachbereich: Informatik
Verfasser: Thomas Ernst
Betreuer: Prof. Jürgen Sauer

Erklärung

1. Mir ist bekannt, dass die Diplomarbeit als Prüfungsleistung in das Eigentum des Freistaats Bayern übergeht. Hiermit erkläre ich mein Einverständnis, dass die Fachhochschule Regensburg diese Prüfungsleistung die Studenten der Fachhochschule Regensburg einsehen lassen darf, und dass sie die Abschlussarbeit unter Nennung meines Namens als Urheber veröffentlichen darf.

2. Ich erkläre hiermit, dass ich diese Diplomarbeit selbständig verfasst, noch nicht anderweitig für andere Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den 05. Mai 2004

.....
Unterschrift

Vorwort

Es gibt ein schier unbegrenztes Angebot an Themen zur Behandlung im Virtuellen Unix Labor[1]. Neben der Integration und Anpassung neuer (Hard- und Software) Komponenten und der Suche und Beseitigung von Sicherheitslücken, oder neben dem Versuch die Performance am Arbeitsplatzrechner oder im Netz ständig zu verbessern, gibt es wohl noch so manches Thema, das bei der täglichen Arbeit eines Systemadministrators oder eines Computeranwenders wichtig ist.

Diese Themen können allerdings hier nicht alle behandelt werden. Einmal sind keine Veränderungen an der Laborhardware möglich, weil die Studenten keinen physischen Zugriff darauf haben. Außerdem stand mir auch nur ein begrenzter zeitlicher Rahmen für die Umsetzung meiner Arbeit zur Verfügung. Für die Übungsdefinition und die zugehörige Ergebnisverifikation ist oft ein detailliertes Hintergrundwissen nötig. Dieses Wissen kann aber nicht für jede Thematik aus allen Bereichen der Informatik vorausgesetzt werden. Ich musste meine Zeit also gut für die Themenrecherche und die damit verbundene Einarbeitung nutzen.

Bei der Einschränkung auf für das Virtuelle Unix Labor relevante Themen, und auch sonst bei so manchem Problem, stand mir Hubert Feyrer ständig zur Seite. Dafür möchte ich mich hiermit bedanken.

[1] Eine nähere Beschreibung des Virtuellen Unix Labors, über dessen Sinn, Aufbau und Technik, finden Sie im Kapitel 1.1 Das Virtuelle Unix Labor (Seite 6).

Inhaltsverzeichnis

Kapitel 1
Einführung

1.1 Das Virtuelle Unix Labor

Das Informatikstudium an der Fachhochschule Regensburg beinhaltet die Pflichtvorlesung „Systemverwaltung unter Unix (SA)“. Bisher standen für Übungen dazu nur die Rechner im Unixlabor zur Verfügung. Diese Rechner werden allerdings auch für andere Veranstaltungen genutzt. Daher ist es nicht möglich, sie vom Rest des Rechennetzes der Fachhochschule abzutrennen. Fehler bei der Konfiguration der Computer, die beim Arbeiten am System immer wieder vorkommen, würden dann aber zu Problemen und Systemausfällen führen, sowie als Sicherheitslücke zum Einfall in das FH Rechensystem ausgenutzt werden können. Daher kann den Studenten kein Rootzugang auf diese Rechner gegeben werden. Systemverwaltung ohne Rootberechtigung ist aber praktisch unmöglich.

Darum wurde das Virtuelle Unix Labor (VULab) entwickelt. Den Studenten wird die Möglichkeit gegeben an einem Netzwerk, das vom allgemeinen Netz der Fachhochschule getrennt ist, zu üben und zu tüfteln. So entstehen bei Fehlern keine negativen Folgen für den laufenden Produktivbetrieb.

Das VULab-Netzwerk besteht derzeit aus drei Rechnern:

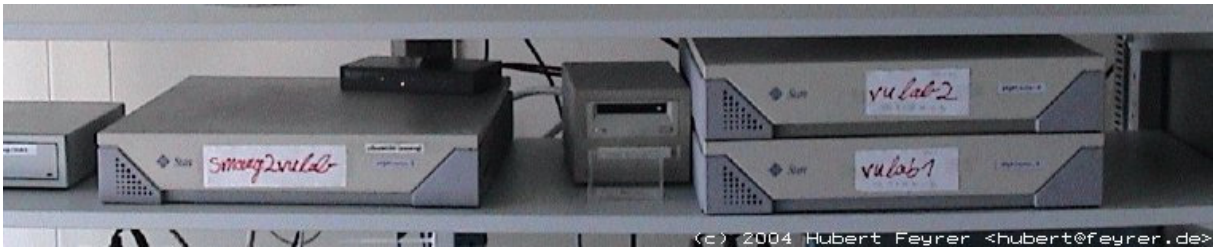

Abbildung 1.1: Die Rechner im VULab

smaug:
NetBSD Laborserver auf einer Sun SPARCStation 5. Dieser Rechner übernimmt die Übungssteuerung und –verwaltung. Er regelt den Zugang zu den restlichen Kursrechnern. Um diese Arbeit leisten zu können, sind folgende Komponenten installiert:

- Apache Weberver mit PHP
- PostgrSQL-Datenbank
- IPfilter Firewall
- DHCP Server

vulab1/vulab2:
Die beiden Kursrechner laufen jeweils auf einer Sun SPARCStation 4. Die zu verwenden Betriebssysteme und die dazugehörigen Konfigurationen sind abhängig von den einzelnen Übungen. Die Rechner können über Telnet, FTP und SSH von den Studenten aus dem Internet angesprochen werden.

Die Übungsbearbeitung läuft im VULab automatisiert ab. Die einzelnen Übungen können innerhalb eines Zeitrasters gebucht werden. Vor Beginn werden die für das jeweilige Szenario nötigen Diskimages an den Kursrechnern installiert, so hat jeder Übungsteilnehmer die gleiche Ausgangssituation. Die Firewall auf dem Verwaltungsrechner sorgt dafür, dass nur der Rechner, an dem der aktuell Übende sitzt, Zugriff auf die Kursrechner bekommt. Dem Studenten steht dann eine bestimmte Bearbeitungszeit zur Verfügung in der er alle Anforderungen erfüllen muss. Nach Übungsende erfolgt eine ebenfalls vom Szenario abhängige Ergebnisverifikation.

Weitere Informationen zum Aufbau, zur Funktion und zu der Technik des Virtuellen Unix Labors finden Sie unter http://www.feyrer.de/vulab/.

1.2 Allgemeines

1.2.1 Erläuterungen zur Themenliste

Ich sollte mit dieser Arbeit möglichst viele Übungsvorschläge für Szenarien im VULab erarbeiten und damit die unterschiedlichsten Betätigungsfelder aus der Informatik erschließen. Dabei sollte ich mich nicht ausschließlich auf Themen der Unix-Welt festlegen und so bei einer späteren Erweiterung auf andere Betriebssysteme auch dafür interessante Übungsbereiche liefern.

Die thematisch sortierte Auflistung aus dem Kapitel 2 stellt das Ergebnis meiner Suche dar. Aufgrund der Fülle an möglichen Themen werde ich nur eine Übung – und zwar zum Thema NIS (Kapitel 2.2.1.1 und 2.2.1.2) - bis hin zur Formulierung der Check-Scripten ausarbeiten. Für die restlichen Themen gebe ich nur kurze Anregungen zu Übungsszenarien, Aufgabenstellungen sowie Möglichkeiten zur bzw. Besonderheiten bei der Ergebnisverifikation. Da im Labor derzeit nur Rechner mit Unix-Systemen zur Verfügung stehen, werden meine Anregungen oft unixspezifisch sein.

1.2.2 Aufbau der einzelnen Übungen

Meine Übungsvorschläge werden wie folgt definiert:

- Beschreibung der Szenarioumgebung
- Angabe der Aufgabenstellung
- Möglichkeiten zur Ergebnisverifikation

1.2.3 Ablauf der Ergebnisverifikation

Die Erfolgskontrolle nach dem Ende der Übungsbearbeitung erfolgt automatisch durch den übergeordneten Steuerungsrechner. So genannte Check-Scripten werden abhängig von den einzelnen Übungen durchlaufen und tragen die Ergebnisse zur weiteren statistischen Auswertung in die Datenbank des Systems ein. Die Scripten werden via Remote Shell auf den Übungsrechnern ausgeführt. Daher kamen zunächst entweder Bourne-Shell- oder Perl-Scripten zur Programmierung der Check-Scripten in Frage. Da das VULab später eventuell noch auf andere Betriebssysteme erweitert werden soll, haben wir – Herr Hubert Feyrer und ich – uns aus Kompatibilitätsgründen für die Verwendung von Perl entschieden.

Bei der Implementierung der ersten Übungen wurde auffällig, dass die Möglichkeit bestehen muss den Script-Ablauf zu manipulieren. Eine Vielzahl ähnlicher Scripten können so durch ein allgemeines ersetzt werden. Die Variation im Programmablauf wird durch das Setzten von globalen Variablen erreicht. Diese können bei manchen Scripten schon mit Standardwerten vorbelegt sein. Daraus ergab sich für die Check-Scripten das Skelett aus Abbildung 1.2.3.

Alle Scripten zur Auswertung bieten folgende Parameter an:

whatis	Eine Kurzbeschreibung des jeweiligen Scripts wird ausgegeben
listparms	Eine Liste aller Variablen mit Defaultwerten und einer Variablenbeschreibung wird ausgegeben
-h	Alle verfügbaren Parameter (also whatis, etc.) werden ausgegeben

Beim Aufruf ohne Parameter wird wie nicht anders zu erwarten die eigentliche Ergebnisverifikation ausgeführt. Dabei erfolgt in der Funktion „init()“ zunächst die Definition der Variablen mit den globalen Werten. Wenn keine entsprechenden globalen Variablen vorhanden sind, werden automatisch die Defaultwerte aus dem Script übernommen. In der Funktion „check()“ finden schließlich die nötigen Abfragen zur Überprüfung statt. Die Scripten geben bei einem richtigen Ergebnis „ok“ auf die Standartdausgabe aus, alle anderen Ausgaben werden vom Steuerungsrechner als Fehler interpretiert.

```
#!/usr/bin/perl
##############################################################################
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Script Kurzbeschreibung
***
';

##############################################################################
# Parameter:
# [ "Variable", "Default, "Beschreibung der Variable" ]
#
@vars = (
  …
);

##############################################################################
# Check-Spezifisch: if true return ok
sub check()
{
  …
}

##############################################################################
### Common code:

# Variablen übernehmen
sub init()
{
  for($i=0; $i<=$#vars; $i++) {
    if(exists($ENV{$vars[$i][0]})) {
      ${$vars[$i][0]} = $ENV{$vars[$i][0]};
    } else {
      ${$vars[$i][0]} = $vars[$i][1];
    }
  }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
  for($i=0;$i<=$#vars;$i++) {
    print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
  }
} elsif($ARGV[0] eq "whatis") {
  $WHATIS=~s/^\n*//g;
  $WHATIS=~s/\n\*\*\* ?//g;
  $WHATIS=~s/^\*\*\* ?//g;
  $WHATIS=~s/\n*$//g;
  print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
  print "whatis     Kurzbeschreibung des Scripts\n";
  print "listparms  Listet Variablen mit Default und Beschreibung\n";
  print "-h         Alle Parameter\n";
  print "sonst      Check-Script wird ausgefuehrt\n";
} else {
  init();
  check();
}
```

Abbildung 1.2.3: Struktur der Check-Scripten

Kapitel 2
Themenliste

Im Anschluss folgen die Ergebnisse meiner hauptsächlichen Aufgabe, nämlich dem Finden von möglichen Übungsszenarien. Ich habe oft nur wenige Übungsdefinitionen innerhalb eines Themenkomplexes explizit ausgearbeitet. Wenn ich z.B. eine Aufgabe für ein bestimmtes Betriebssystem mit einem bestimmten Servertyp formuliert habe, kann diese Übung meist so oder so ähnlich auch mit einer anderen Übungsumgebung durchgeführt werden. Das Bedeutet, dass sich aus einem Übungsszenario und den anschließenden Informationsquellen eine Vielzahl zusätzlicher Szenarien ergeben, die sich jeweils nur wenig unterscheiden.

Ich habe viele der von mir definierten Aufgaben eigenhändig ausprobiert und so versucht, Fehler und Probleme beim späteren Übungssetup zu vermeiden. Es gab jedoch diverse Gründe, warum ich nicht alle Schwierigkeiten ausschließen kann. Einmal stand mir nicht die gleiche Ausstattung wie im tatsächlichen Labor zur Verfügung. Die jeweiligen Testinstallationen habe ich auf PC-Systemen durchgeführt, da ich keinen Zugriff auf SPARCStations, die ja im Labor verwendet werden, hatte. Außerdem hatte ich nur administrativen Zugriff auf Linux und Windows XP Rechner. Manche Szenarien konnte ich daher nur theoretisch erarbeiten, sie sollten trotzdem ohne große Anpassungsarbeit durchführbar sein. Es wäre zeitlich sowieso nicht möglich gewesen, jede Software auf jeder denkbaren Hardwareumgebung zu testen. Anpassungen bei der endgültigen Aufgabendefinition sind also nicht zu verhindern.

2.1 Web

2.1.1 Apache Webserver aufsetzen

Szenario

Für diese Übung wird nur einer der Kursrechner benötigt. Auf ihm ist ein beliebiges Unix-System ohne besondere Erweiterungen installiert.

Ein aktuelles Apache-Quellpaket muss den Übenden zur Verfügung stehen. Entweder man gibt es ihnen vor (wie hier unter /root/src/http-2.0.48.tar.gz) oder man legt fest wo und in welcher Version es bezogen werden soll.

Aufgabe

Installieren Sie auf dem Rechner vulab1 einen Apache Webserver.

Das dazu nötige Quellpaket liegt unter /root/src/httpd-2.0.48.tar.gz. Der Server soll im Verzeichnis /usr/local/apache2 arbeiten, auf dem Standardport 80 horchen und nach jedem Reboot erneut gestartet werden.

Überprüfung

Zur Verifikation kann plattformunabhängig überprüft werden, ob der Webserver eine .html-Datei aus dem Datenverzeichnis korrekt ausliefert, z.B. über den Aufruf „GET <Serveradresse>/index.html“. Dabei steht <Serveradresse> für den Rechnernamen oder die entsprechende IP-Adresse des Webservers.

2.1.2 Apache mit PHP

Webserver werden inzwischen nicht mehr nur zum Ausliefern von .html-Dateien benutzt. Ein moderner Webserver soll häufig mit einigen der folgenden Erweiterungen betrieben werden können:

- Die sichere Datenübertragung über Secured Socket Layer (SSL) oder Transport Layer Security (TLS).

- Das Einbinden und Ausführen von Scripten zur dynamischen Auswertung von Benutzereingaben. Hier eine Auswahl an Scriptsprachen: Active Server Pages (ASP), Java Server Pages (JSP), PHP: Hypertext Preprocessor (PHP)

- Die Bereitstellung einer Schnittstelle zur Ausführung von Programmen verschiedener Sprachen. Dies wird über das Common Gateway Interface (CGI) erreicht, wodurch vor allem Perl, C und einfache Bourne-Shell Scripte zur Bearbeitung genutzt werden können.

Der Apache Webserver basiert auf einem Modulkonzept. Viele der oben genannten Erweiterungen werden durch das Einbinden eines entsprechenden Moduls erreicht. Manchmal muss allerdings zusätzliche Software installiert werden. Zur Unterstützung von SSL-Verschlüsselung muss beispielsweise ein Paket wie etwa OpenSSL integriert sein. Manche Webserver haben einige zusätzliche Features bereits eingebaut. Der Microsoft Internet Information Server etwa beherrscht bereits die Auswertung von .asp-Dateien.

Aus diesem Bereich können beliebige Kombinationen aus Webserver und Zusatzfeature zum Übungsinhalt gemacht werden.

Szenario

Für diese Übung wird nur einer der Kursrechner benötigt, auf ihm muss ein beliebiges Unix-System ohne besondere Erweiterungen installiert sein. Die Sourcen für einen PHP-Interpreter und einen Apache Webserver werden zusammen mit einer Testdatei datedemo.php vorgegeben. Diese Datei liefert die aktuelle Uhrzeit des Systems zurück. Über die Uhrzeit kann bestimmt werden, ob wirklich das Script ausgeführt wird, oder ob eine „gefälschte“ Datei mit einem statischen Eintrag eine richtige Antwort vortäuschen soll. Um den Arbeits- und Formatierungsaufwand gering zu halten, empfehle ich die Verwendung der PHP Funktion time() in der Datei datedemo.php. So kann der zurückgegebene Unix-Zeitstempel direkt mit der Systemzeit verglichen werden, ohne dazwischen mehrmals von einem Format in das andere umgerechnet werden zu müssen.

Aufgabe

Auf dem Rechner vulab1 liegt unter /root/src/ die Datei datedemo.php. Diese Datei soll über einen Apache Webserver bereitgestellt werden.

1. Installieren sie dazu auf dem Rechner vulab1 einen PHP-fähigen Apache Webserver. Die zu verwendenden Quellpakete liegen unter /root/src/.

2. Kopieren sie die Datei datedemo.php in das Datenverzeichnis des Servers und überprüfen sie, ob bei einem Aufruf des Scripts die aktuelle Uhrzeit ausgegeben wird.

Alle Änderungen sollen nach einem Neustart noch aktiv sein.

Überprüfung

Die Verifikation kann wieder über den GET-Aufruf stattfinden. Liefert „GET <Serveradresse>/datedemo.php“ die aktuelle Uhrzeit und nicht den Programmquellcode zurück, arbeitet der Server richtig.

2.1.3 Logfile Analyse / Serverstatistik

Informationen über die Aktionen auf einem Webservers werden in Logdateien gesichert. Zur einfacheren Informationsgewinnung gibt es eine Reihe von Analysetools, die die Inhalte der Logdateien visuell darstellen. Viele dieser Programme können auch zur Systemanalyse andere Dienste wie etwa FTP verwendet werden.

Szenario

Benötigt wird ein Rechner mit einem Unix-System und lauffähigem Webserver. Unter /var/log/httpd/access_log liegt ein vorbereitetes Logfile (z.B. eines Apache Webserver). Unter /root/src liegt die Tarball webalizer-2.01-10-src.tgz für das Statistiktool Webalizer.

Aufgabe

Auf dem Rechner vulab1 liegt unter /var/log/httpd/ die Datei access_log. Entnehmen sie dieser Datei, wie viele kB im Monat Februar 2004 von der IP-Adresse 132.199.1.1 abgerufen wurden.

Gehen sie dazu so vor:
1. Insallieren sie das Statistiktool Webalizer. (Source unter /root/src/).
2. Erzeugen sie die Statistikausgabe im Verzeichnis /usr/local/apache2/htdocs/statistik/.
3. Entnehmen Sie der Statistik unter http://servername/statistik den geforderten Wert.
4. Erzeugen sie im Verzeichnis /root/src/ die Datei res.txt mit dem kB-Wert als Inhalt.

Überprüfung

Es kann überprüft werden, ob in der Datei /root/src/res.txt das richtige Ergebnis steht. Wenn die Studenten gut zusammenarbeiten, wird dieser Wert allerdings untereinander weitergegeben werden. Deshalb sollte man in den Logfiles des aktuell laufenden Webservers prüfen, ob eine der Aufgabenstellung entsprechende URL während der Bearbeitungszeit angesurft wurde.

2.1.4 Informationsquellen

Verschiedene Webserver

- AOLserver, Webserver für Unix
  im Internet unter http://www.aolserver.com/

- Apache, freier Webserver für Unix und Windows
  im Internet unter http://www.apache.org/ oder http://www.apachefriends.org/

- Internet Information Server, Webserver für Windows
  im Internet unter http://www.microsoft.com/

- Sun One Webserver (früher iPlanet), Webserver für Unix und Windows
  im Internet unter http://de.sun.com/Produkte/software/index.html

- TUX, Kernel-Webserver unter Linux
  im Internet unter http://www.redhat.com/docs/manuals/tux/TUX-2.0-Manual/

- Zope, Webanwendungserver für Unix und Windows
  im Internet unter http://www.zope.org/ oder http://www.dzug.org/

Verschiedene Analysetools

- Analog, für Unix, Windows und MacOS
  im Internet unter http://www.analog.cx/

- LFApro, für Windows
  im Internet unter /www.lfa-pro.de/

- Lire, für Unix und MacOS
  im Internet unter http://logreport.org/lire/

- Webalizer, für Unix, Windows und MacOS
  im Internet unter http://www.webalizer.org/

Weitere Quellen

- Ein deutsches ASP-Forum
  im Internet unter http://www.aspgerman.com/aspgerman/

- Enhydra, ein freier Servlet-Container für JSP
  im Internet unter http://www.enhydra.org/

- jetty, ein freier Servlet-Container für JSP
  im Internet unter http://jetty.mortbay.com/jetty/index.html

- jo!, ein freier Servlet-Container für JSP
  im Internet unter http://www.tagtraum.com/

- OpenSSL, freie Implementierung von SSL und TLS
  im Internet unter http://www.openssl.org/

- PHP
  im Internet unter http://www.php.net/

- Produktbeschreibungen zu JSP der Firma Sun
  im Internet unter http://java.sun.com/products/jsp/index.jsp

- Tomcat, ein Servlet-Container für folgende Erweiterungen: SSI, SSL, CGI, JSP
  im Internet unter http://jakarta.apache.org/tomcat/index.html

Verschiedene Webbrowser

- Amaya; für Windows, Unix und MacOS
  im Internet unter http://www.w3.org/Amaya/

- Beonex, für Windows, Unix und MacOS
  im Internet unter http://www.beonex.com/

- Camino, für MacOS
  im Internet unter http://www.mozilla.org/projects/camino/

- Crazy Browser, für Windows
  im Internet unter http://www.crazybrowser.com/

- Dillo, für Unix
  im Internet unter http://www.dillo.org/

- Epiphany, für Unix
  im Internet unter http://www.gnome.org/projects/epiphany/

- Galeon, für Unix
  im Internet unter http://galeon.sourceforge.net/

- iCab, für MacOS
  im Internet unter http://www.icab.de/

- K-Meleon, für Windows
  im Internet unter http://en.wikipedia.org/wiki/K-Meleon

- Konqueror, für Unix
  im Internet unter http://www.konqueror.org/

- Links, für Unix
  im Internet unter http://artax.karlin.mff.cuni.cz/~mikulas/links/

- Lynx, für Windows und Unix
  im Internet unter http://lynx.browser.org/

- Mozilla, für Windows, Unix und MacOS
  im Internet unter http://www.mozilla.org/

- MS Internet Explorter, für Windows und MacOS
  im Internet unter http://www.microsoft.com/germany/ms/internet/default.htm

- Netscape Navigator, für Windows und Unix
  im Internet unter http://channels.netscape.com/ns/browsers/default.jsp

- Omniweb, für MacOS
  im Internet unter http://www.omnigroup.com/applications/omniweb/

- Opera, für Windows, Unix und MacOS
  im Internet unter http://www.opera.com/

- Safari, für MacOS
  im Internet unter http://www.apple.com/de/safari/

- Salamander, für Unix
  im Internet unter http://members.tripod.com/OskarK/salamander.html

- Skipstone, für Unix
  im Internet unter http://www.muhri.net/skipstone/

- w3m, für Unix
  im Internet unter http://w3m.sourceforge.net/index.en.html

2.2 Informationsdienste

2.2.1 Verzeichnisdienste

Im Folgenden werden Verzeichnisdienste – im englischen Directory Services (DS) – behandelt. Über sie werden Daten für die unterschiedlichsten Anwendungen im Netz bereitgestellt.

Wie bereits in der Einführung erwähnt, sind die Kapitel 2.2.1.1 und 2.2.1.2 komplett zu Übungsvorlagen (mit Check-Scripten) ausgearbeitet. Meine Anregungen darin wurden bereits so ähnlich für einen Übungssetup verwendet, der sich bislang bei Tests erfolgreich bewährt.

2.2.1.1 NIS-Master aufsetzen

Szenario

Auf beiden Kursrechnern ist jeweils ein funktionierendes Unix-System ohne NIS installiert. Der für NIS notwendige Code ist unter den Standardpfaden vorhanden. Auf dem Rechner vulab1 existiert ein Account „test“ mit dem Passwort „test“.

Aufgabe

NIS-Master aufsetzen

1. Setzen sie auf dem Rechner vulab1 einen NIS-Master auf.
Der Master soll die Benutzerinformationen von vulab1 im Labornetz bereitstellen und diese in einer Passwortdatei unter /var/yp führen.

Verwenden Sie den NIS-Domänennamen „mynisdomain“.

2. Stellen Sie zusätzlich den yppasswdd-Dämonen zum Ändern des NIS-Passwortes bereit.

Alle Veränderungen sollen nach einem Neustart noch verfügbar sein.

Überprüfung

Die folgenden Tests sind auf dem Rechner vulab1 zu durchlaufen. Die verwendeten Scripten sind im Anhang A abgedruckt.

1. *NIS-Domainname*
Testen, ob die Unix-Funktion „domainname“ das richtige Ergebnis liefert, dazu:
- PROGRAM = domainname setzen
- OUTPUT_SHOULD = mynisdomain setzen
- check-program-output (Anhang A; Seite 48) ausführen

2. *NIS-Domainname in /etc/defaultdomain*
Den Inhalt der Datei überprüfen, dazu:
- FILENAME = /etc/defaultdomain setzen
- CONTENT_SHOULD = mynisdomain setzen
- check-file-contains (Anhang A; Seite 44) ausführen

3. *Existiert /var/yp/passwd* ?
Dazu:
- FILENAME = /var/yp/passwd setzen
- check-file-exists (Anhang A; Seite 46) ausführen

4. *Einträge von /var/yp/passwd*
Prüfen, ob diese Datei sinnvolle Einträge für eine Passwortdatei hat, dazu:
- FILENAME = /var/yp/passwd setzen
- CONTENT_SHOULD = ^vulab: setzen
- check-file-contains (Anhang A; Seite 44) ausführen

5. *Existiert /var/yp/Makefile* ?
Dazu:
- FILENAME = /var/yp/Makefile setzen
- check-file-exists (Anhang A; Seite 46) ausführen

6. *Einträge von /var/yp/Makefile*
Prüfen, ob diese Datei sinnvolle Einträge als NIS-Makefile hat, dazu:
- FILENAME = /var/yp/Makefile setzen
- CONTENT_SHOULD = ^NOPUSH setzen
- check-file-contains (Anhang A; Seite 44) ausführen

7. *RPC-Dienst ypserv*
Prüfen, ob ypserv registriert wurde, dazu:
- PROGRAM_NR = 100004 setzen
- check-rpc-reg (Anhang A; Seite 50) ausführen

8. *RPC-Dienst yppasswdd*
Prüfen, ob yppasswdd registriert wurde, dazu:
- PROGRAM_NR = 100009 setzen
- check-rpc-reg (Anhang A; Seite 50) ausführen

9. *ypserv beim reboot*
Wird ypserv beim reboot gestartet (NetBSD spezifisch)?
Gibt es ein Startscript in /etc/rc.d/, dazu:
- FILENAME=/etc/rc.d/*ypserv* setzen
- check-file-exists (Anhang A; Seite 46) ausführen

10. *yppasswdd beim reboot*
Wird yppasswdd beim reboot gestartet (NetBSD spezifisch)?
Gibt es ein Startscript in /etc/rc.d/, dazu:
- FILENAME=/etc/rc.d/*yppasswdd* setzen
- check-file-exists (Anhang A; Seite 46) ausführen

2.2.1.2 NIS-Client aufsetzen

Szenario

Die NIS-Master- und –Client-Übung sollten zusammen bearbeitet werden. Das Szenario aus 2.2.1.1 ist also unverändert.

Zur Überprüfung, ob das Passwort wirklich über passwd geändert wurde, liegt auf dem Server eine „gefälschte“ passwd-Datei unter /usr/bin/. Wenn die Studenten das passwd-Kommando ausführen, wird so nicht die ursprüngliche Funktion zum Ändern des Unix Passwortes aufgerufen, sondern die manipulierte Datei. Diese führt das ursprüngliche passwd-Kommando aus und protokolliert anschließend in der Datei /var/log/passwd.log, ob das Passwort der Aufgabenstellung entsprechend geändert wurde. Eine Vorlage für die genannte passwd ist im Anhang A abgedruckt und sollte noch an das jeweilige System angepasst werden. Die „echte“ passwd muss für mein Szenario in dwssap (passwd rückwärts) umbenannt werden.

Aufgabe

NIS-Client aufsetzen

1. Setzen Sie auf dem Rechner vulab2 einen NIS-Client auf, der die Informationen vom NIS-Master auf vulab1 bezieht. Ändern Sie anschließend unter Verwendung des passwd-Kommandos das Passwort des Nutzers „test“ von „test“ auf „neues“.

Überprüfung

Die folgenden Tests sind auf dem Rechner vulab2 zu durchlaufen. Die verwendeten Scripten sind im Anhang A abgedruckt.

1. *NIS-Domainname*
Testen, ob die Unix-Funktion „domainname“ das richtige Ergebnis liefert, dazu:
- PROGRAM = domainname setzen
- OUTPUT_SHOULD = mynisdomain setzen
- check-program-output (Anhang A; Seite 48) ausführen

2. *NIS-Domainname in /etc/defaultdomain*
Den Inhalt der Datei überprüfen, dazu:
- FILENAME = /etc/defaultdomain setzen
- CONTENT_SHOULD = mynisdomain setzen
- check-file-contains (Anhang A; Seite 44) ausführen

3. *RPC-Dienst ypbind*
Prüfen, ob ypserv registriert wurde, dazu:
- PROGRAM_NR = 100007
- check-rpc-reg (Anhang A; Seite 50) ausführen

4. *Einträge von /etc/nsswitch.conf*
Prüfen, ob NIS für Benutzerinformationen ausgewertet wird, dazu:
- FILENAME = /etc/nsswitch.conf setzen
- CONTENT_SHOULD =
  ^[[:space:]]*(passwd){1}[a-zA-Z:[:space:]]*(nis){1,} setzen
- check-file-contains (Anhang A; Seite 44) ausführen

5. *Account via ypcat passwd*
Prüfen, ob ein Benuzter in ypcat passwd bekannt ist, dazu:
- USER_SHOULD = ^vulab setzen
- nis-check-ypcat (Anhang A; Seite 51) ausführen

6. *Account via finger*
Prüfen, ob ein Benuzter per finger bekannt ist, dazu:
- USER_SHOULD = vulab setzen
- check-finger (Anhang A; Seite 47) ausführen

7. *neues Passwort*
Prüfen, ob das Passwort für den Account „test“ auf „neues“ gesetzt wurde, dazu:
- FILENAME = /var/log/passwd.log setzen
- CONTENT_SHOULD = test->neues; ok setzen
- check-file-contains (Anhang A; Seite 44) ausführen

2.2.1.3 OpenLDAP-Adressbuch

Szenario

Auch für diese Übung wird nur einer der Kursrechner benötigt auf dem ein Unix-System installiert sein muss. Auf ihm soll ein Server eingerichtet werden, der ein Adressbuch verwaltet und über LDAP verbreitet. Das OpenLDAP-Paket liegt im Verzeichnis /root/src mit einer vorbereiteten Konfigurationsdatei sldap.conf (Anhang A; Seite 53), damit eine einheitliche Administratorkennung zur Verifikation benutzt werden kann. In diesem Verzeichnis kann den Studenten auch eine .ldif-Datei (adressen.ldif: Anhang A; Seite 44) mit Adressdaten hinterlegt werden, die nur noch in den Verzeichnisdienst eingepflegt werden muss. Es kann aber auch gefordert werden, dass diese Datei selbst erstellt werden soll.

Zusätzlich könnte auf dem zweiten Rechner ein LDAP-Slave aufgesetzt werden, der ebenso die Daten ausliefert. Dann müssen dort ebenso die Quellen verfügbar sein.

Aufgabe

Installieren sie auf dem Rechner vulab1 einen LDAP-Server, der Adressdaten aufnehmen soll.

Im Verzeichnis /root/src/ liegen die nötigen Quellen zum Aufbau des Dienstes und die Datei adressen.ldif mit den zu verwendenden Datensätzen. Dort befindet sich auch die zu verwendende Konfigurationsdatei sldap.conf.

Überprüfung

Mit dem Kommando „ldapadd“ sollte versucht werden, ob der Server zusätzliche Daten aufnimmt, die dann auch über „ldapsearch“ wieder ausgeliefert werden.

Bei der Master/Slave-Variante muss die Abfrage auch an den Slave gerichtet werden. Dabei ist zu beachten, dass ausreichend mit der Abfrage gewartet wird, um den beiden Rechnern die Möglichkeit zu geben die Daten abzugleichen. Die Zeitspanne zwischen der Replizierung der Informationen muss sinnvoll klein gewählt sein und sollte deshalb bereits in der Aufgabenstellung vorgegeben werden.

2.2.1.4 Informationsquellen

Verschiedene Implementierungen von Verzeichnisdiensten

- Active Directory (ADS) der Firma Microsoft
  im Internet unter
  http://www.microsoft.com/windows2000/technologies/directory/ad/default.asp/

- Network Information Service (NIS) der Firma Sun

- Novell Directory Services (NDS) der Firma Novell

- Open Directory der Firma Apple

- OpenLDAP, GPL Verzeichnisdienst
  im Internet unter http://www.openldap.org/

- Oracle Internet Directory der Firma Oracle

Weiterführende Links

- LDAP Artikel- und Whitpapersammlung
  im Internet unter http://www.bind9.net/ldap/

2.2.2 Domain Name Service (DNS)

2.2.2.1 DNS-Server mit BIND

Szenario

In dieser Übung soll auf vulab1 ein DNS-Server aufgesetzt werden. Der Rechner muss mit einem Unix-System und den BIND-Quellen vorbereitet werden. Der zweite Rechner wird am Ende zur Verifikation benutzt. Auf ihm muss ein beliebiges Betriebssystem laufen. Er bekommt eine feste IP-Adresse zugewiesen, hier im Beispiel 10.0.0.1.

Aufgabe

Installieren sie auf dem Rechner vulab1 einen DNS-Server für die Zone „vulab“.

Setzen sie den Server auf und generieren sie die nötigen Zonendateien. Tragen sie für die Zone „vulab“ den Rechner „test.vulab“ mit der IP-Adresse 10.0.0.1 ein.
Der Rechner vulab2 soll diesen DNS-Server zur Namensauflösung benutzen.

Alle Änderungen sollen auch noch einem Neustart von vulab1 noch aktiv sein.

Überprüfung

Es ist zu Prüfen, ob der Name „test.vulab“ über den DNS-Server aufgelöst wird. Dies kann z.B. mit dem Aufruf von „ping test.vulab“ beim Rechner vulab2 geschehen. Wenn die Namensauflösung funktioniert, sollten keine Zeitüberschreitungen auftreten.

2.2.2.2 Informationsquellen

Nameserver Software

- Berkley Internet Name Domain (BIND)
  im Internet unter http://www.isc.org/index.pl?/sw/bind/

- DJBDNS
  im Internet unter http://cr.yp.to/djbdns.html

- MaraDNS
  im Internet unter http://www.maradns.org/

- Name Server Daemon (NSD)
  im Internet unter http://www.nlnetlabs.nl/nsd/index.html

- PowerDNS

Erweiterungen zu DNS

- DNS Security (DNSsec);
  Gesicherte DNS-Kommunikation
  im Internet unter http://www.dnssec.net/

- Dynamic DNS;
  Zuweisung eines Domainnamen zu einer variablen IP-Adresse
  im Internet unter http://www.dyndns.org/

- Electronic Numbering (ENUM);
  Ansprechen von Geräten über das Telefonadressschema

- Internationalizing Domain Names in Applications (IDNA);
  Domainnamen mit Nicht-ASCII-Zeichen

- Transaction Signatures (TSIG);
  Gesicherte DNS-Kommunikation

2.2.3 Dynamic Host Configuration Protocol (DHCP)

2.2.3.1 Serverinstallation

Im VULab beziehen die Kursrechner ihre Adressen bereits über einen DHCP-Server. Diese Konfiguration sollte nicht verändert werden, da die Rechner sonst eventuell bei der Verifikation nicht mehr angesprochen werden können. Deshalb wird es im VuLab wohl keine Übung zu DHCP geben. Generell wären Übungen aus dem Bereich DHCP schon denkbar und interessant. Das folgende Szenario bezieht sich daher nicht auf unsere Laborumgebung.

Szenario

Es werden drei Rechner benötigt, davon einer mit einem Unix-System und den DHCP-Quellen installiert, die anderen mit einem beliebigen Betriebssystem.

Aufgabe

Auf einem Rechner soll der DHCP-Server installiert werden, der der Netzwerkkarte eines Rechners Adressen aus einem dynamischen Adressbereich zuweist. Dieser Bereich sollte möglichst klein (z.B. zwei Adressen) gewählt werden, damit bei der Verifikation nicht so viele Adressen überprüft werden müssen. Der Netzwerkkarte des anderen Rechners soll eine feste IP-Adresse zuordnet werden.

Überprüfung

Es ist zu Prüfen, ob der eine Rechner über die feste IP-Adresse, und der andere über eine der Adressen aus dem dynamischen Bereich angesprochen werden kann.

2.2.3.2 Informationsquellen

Weiterführende Links

- DHCP Artikel- und Whitpapersammlung
  im Internet unter http://www.bind9.net/dhcp/

- DHCP-Resources
  im Internet unter http://www.isc.org/index.pl?/dhcp/

- DHCP-Resources
  im Internet unter http://www.dhcp.org/

2.2.4 Datenbanken

2.2.4.1 Serverinstallation

Es gibt eine Vielzahl von Datenbanksystemen die innerhalb einer Übung installiert werden könnten. Ich habe mich für eine MySQL-Datenbank entschieden, da im Internet häufig die Kombination PHP/MySQL zum Einsatz kommt.

Szenario

Auf vulab1 ist ein beliebiges Betriebssystem installiert und es läuft darauf zusätzlich ein Apache Webserver mit PHP. Auf dem Rechner liegt ein SQL-Script, das mehrere Tabellen mit einer großen Anzahl von Daten füllt. Zusätzlich gibt es ein PHP-Script zur Abfrage der Datenbank. Dieses Script liefert, nachdem man ihm einen Tabellen- und Feldnamen als Parameter mitgegeben hat, den Inhalt des betreffenden Feldes. Bei einer genügend großen Datenbank kann so erreicht werden, dass der Übende nicht die Ausgaben der Datei vortäuschen kann, ohne die Datenbank einzurichten.

Aufgabe

1. Installieren Sie auf dem Rechner vulab1 eine MySQL-Datenbank.
(Beziehen Sie eine aktuelle Version der Software unter http://www.mysql.org/)

2. Legen Sie den Benutzer „dbuser“ mit dem Passwort „dbpw“ an. Erzeugen Sie die Datenbank „testing“, der Benutzer „dbuser“ soll Zugriff auf diese Datenbank haben.

3. Lassen Sie das SQL-Script data.sql auf ihrem System laufen, das eine Reihe von Tabellen auf ihrem System anlegt. (data.sql liegt im Verzeichnis /root/src)

Überprüfung

Über das PHP-Script wird versucht einzelne Felder aus der Datenbank abzufragen. Werden die Daten richtig ausgeliefert, sollte die Datenbank richtig konfiguriert sein.

2.2.4.2 Informationsquellen

Verschiedene Datenbanksysteme

- Firebird (früher InterBase); relationale OpenSource Datenbank
  im Internet unter http://www.firebirdsql.org/ und http://www.firebird-datenbank.de/

- MaxDB (früher SAP DB bzw. Adabas); relationale Datenbank der Firma SAP

- Microsoft SQL-Server (MSSQL); relationale Datenbank der Firma Microsoft
  im Internet unter http://www.microsoft.com/

- MS Access; Datenbank der Firma Microsoft
  im Internet unter http://www.microsoft.com/

- MySQL; relationale OpenSource Datenbank
  im Internet unter http://www.mysql.org/

- Oracle; relationale Datenbank der Firma Oracle
  im Internet unter http://otn.oracle.com/

- PostgreSQL; objektrelationale OpenSource Datenbank
  im Internet unter http://www.postgresql.org/ und http://postgres.de/

- Sybase SQL-Server; relationale Datenbank der Firma Sybase
  im Internet unter http://www.sybase.com/home/

- Tdbengine; freie relationale Datenbank
  im Internet unter http://www.tdbengine.org/

2.3 eMail

2.3.1 Serverinstallation

Szenario

Auf beiden Kursrechnern liegt ein Unix-System vor, bei denen der Account „peter“ (evtl. über NIS) existiert. Am Rechner vulab1 ist bereits das Programm procmail installiert, das die eMails lokal verwaltet und verteilt.

Aufgabe

Installieren sie in der Labor-Domäne „vulab“ ein eMail-System. Führen sie dazu folgende Arbeitsschritte aus:

1. qmail SMTP-Server auf dem Rechner vulab2
Die Nachrichten sollen über den Rechner vulab1 verteilt werden.
(qmail ist unter http://cr.yp.to/qmail.html erhältlich)

2. alias-Definition
Der Benutzer „peter“ soll zusätzlich die Mails der Adresse „info@vulab“ („info“ ist kein Account auf dem System) empfangen.

3. qmail auf dem Rechner vulab1
Installieren Sie hier ebenfalls den qmail-Server zum Umgang mit SMTP-Anfragen. Die Nachrichten der Domäne „vulab“ sollen hierbei lokal über procmail ausgeliefert werden. procmail läuft bereits auf dem System.

4. POP-Server auf vulab1
Verwenden Sie dazu „qmail-pop3d“. Dieser POP3-Server ist im qmail-Paket enthalten, kann aber unabhängig davon installiert werden. Stellen sie sicher, dass am Rechner vulab1 POP-Anfragen entgegengenommen werden.

5. Fetchmail auf dem Rechner vulab2
Installieren Sie Fetchmail auf dem Rechner vulab2, um Nachrichten von vulab1 abzuholen. (Sourcen unter http://catb.org/~esr/fetchmail/)

6. Erzeugen Sie einen Cronjob, der in jeder Minute die Nachrichten an „peter“ von vulab1 nach vulab2 holt.

Eine deutsche Anleitung zur Installation und Konfiguration unter anderem von qmail finden Sie unter http://www.wallroth.de/sebastian/lwq.html

Überprüfung

Mit dem Kommando „qmailctl stat“ auf vulab1 und vulab2 testen, ob die SMTP-Server laufen. Dabei sollte für die vier Services qmail-send, qmail-send/log, qmail-smtpd und qmail-smtpd/log jeweils „up“ angezeigt werden.

Vom Rechner vulab2 aus ein eMail an die Adresse „info@vulab“ senden. Durch ein entsprechendes Checkscript den Kontrollrechner eine Minute lang warten lassen und danach testen, ob bei vulab2 eine Nachricht für den Benutzer „peter“ angekommen ist. In /var/qmail/control/locals sollte überprüft werden, das dieses Mail nicht Lokal auf vulab2 ausgeliefert wurde, sondern den Umweg über vulab1 gemacht hat.

2.3.2 Mailfilter

Szenario

Am Rechner vulab1 läuft auf einem Unixsystem das Programm procmail. Es ist ein Account „peter“ vorhanden. Es gibt eine Datei /home/peter/file.

Aufgabe

Benutzen Sie das Programm procmail als eMail-Filter für den Benutzer „peter“ am Rechner vulab1. Erstellen Sie dazu eine Datei „.procmailrc“ in Homeverzeichnis des Benutzers „peter“, in der die Filterregeln folgendermaßen bestimmt werden sollen:

1. Nachrichten deren Betreff den Begriff „spam“ enthält sollen im Mailfolder „spam“ abgelegt werden.

2. Bei einer eMail vom Benutzer „root“ mit dem Betreff „delete file“ soll die Datei „file“ im Homeverzeichnis des Benutzers „peter“ gelöscht werden.

Überprüfung

Es werden den Regeln entsprechende Mails gesendet. Danach testet man, ob die gewünschten Resultate eingetreten sind.

2.3.3 Mailinglisten

Szenario

Auf dem Rechner vulab1 ist ein vollständiges eMail-System konfiguriert, der majordomo-Tarball liegt unter /root/src. In der Datei /etc/aliases sind eine Reihe von Benutzern beim Aliasnamen „vulab“ eingetragen.

Aufgabe

In der Datei „/etc/aliases“ ist über den Aliasnamen „vulab“ eine einfache Mailingliste realisiert. Diese Lösung ist jedoch wenig komfortabel, da diese Liste von einem Systemverwalter gepflegt werden muss und sich so sich beispielsweise nicht Nutzer selbst für die Liste anmelden können. Ein Tool das die Verwaltung einer solchen Mailingliste übernimmt, ist z.B. majordomo. Lassen sie also die Liste vulab@vulab1 über majordomo verwalten (Source unter /etc/src). Anmeldungen sollen über eMails mit dem Inhalt „subscribe vulab“ durchgeführt werden, Abmeldungen über „unsubscribe vulab“.

Überprüfung

Zur Überprüfen ist, ob man sich per eMail bei der Mailingliste anmelden kann, und danach die Nachrichten an diese Liste auch erhält. Bei der Anmeldung wird im Normalfall die eMail-Adresse auf eine korrekte Syntax geprüft. Demzufolge wären Adressen der vorm name@vulab falsch, weil kein Punkt nach dem „@“-Zeichen vorhanden ist. Diese Überprüfung muss also von den Übungsteilnehmern (durch eine Änderung in dem majordomo-Script) deaktiviert werden.

2.3.4 Informationsquellen

Verschiedene Mail Delivery Agents (MDA)

- maildrop
  im Internet unter http://www.flounder.net/~mrsam/maildrop/index.html

- procmail
  im Internet unter http://www.procmail.org/

Verschiedene Mail Transfer Agents (MTA)

- Courier; für Unix
  im Internet unter http://www.courier-mta.org/

- Exim; für Unix
  im Internet unter http://www.exim.org/

- MS Exchange Server; für Windows
  im Internet unter http://www.microsoft.com/exchange/

- Postfix, für Unix
  im Internet unter http://www.postfix.org/

- qmail; für Unix
  im Internet unter http://www.qmail.org/

- sendmail; für Unix
  im Internet unter http://www.sendmail.org/

Verschiedene Mail User Agents (MUA)

- Apple Mail; eMailclient der Firma Apple Computer, für MacOS
  im Internet unter http://www.apple.com/macosx/features/mail/

- Bloomba; eMailclient der Firma Stata Labs, für Windows
  im Internet unter http://www.statalabs.com/

- Elm; freier textbasierter eMailclient, für Unix
  im Internet unter http://www.instinct.org/elm/

- Elmo; freier eMailclient, für Unix
  im Internet unter http://elmo.sourceforge.net/

- Eudora; freier eMailclient, für Windows, MacOS und PalmOS
  im Internet unter http://www.eudora.com/

- Gnus; freier Mail- und Newsreader, für Windows, Unix und MacOS
  im Internet unter http://www.gnus.org/

- Incredimail; freier eMailclient, für Windows
  im Internet unter http://www.incredimail.com/

- KMail; freier eMailclient, für Unix
  im Internet unter http://kmail.kde.org/

- MS Outlook / Outlook Express;
  Mail- , Newsreader bzw. Groupware der Firma Microsoft für Windows und MacOS
  im Internet unter http://www.microsoft.com/

- Mutt; freier textbasierter eMailclient, für Unix
  im Internet unter http://www.mutt.org/

- Pegasus Mail; freier eMailclient, für Windows
  im Internet unter http://www.pmail.com/

- Pine; freier eMailclient, für Windows, Unix und MacOS
  im Internet unter http://www.washington.edu/pine/

- Sylpheed; freier Mail- und Newsreader, für Unix
  im Internet unter http://sylpheed.good-day.net/

- The Bat!; eMailclient der Firma RITLABS, für Windows
  im Internet unter http://www.ritlabs.com/

- Thunderbird (füher Minotaur); freier eMailclient, für Windows, Unix und MacOS
  im Internet unter http://www.mozilla.org/products/thunderbird/

- Ximian Evolution; freier eMailclient, für Unix
  im Internet unter http://www.ximian.com/products/evolution/

Mailfiltersoftware

- Anti-Spam SMTP-Proxy (ASSP)
  im Internet unter http://assp.sourceforge.net/

- Bogofilter
  im Internet unter http://bogofilter.sourceforge.net/

- SpamAssassin
  im Internet unter http://eu.spamassassin.org/

- SpamBayes
  im Internet unter http://spambayes.sourceforge.net/

- Spamihilator
  im Internet unter http://www.spamihilator.com/

- SpamPal
  im Internet unter http://www.spampal.org/

- Tagged Message Delivery Agent (TMDA)
  im Internet unter http://tmda.net/

- Vipul's Razor
  im Internet unter http://razor.sourceforge.net/

Mailinglistensoftware

- ezmlm/idx

- Majordomo
  im Internet unter http://www.greatcircle.com/majordomo/

- Mailman
  im Internet unter http://www.list.org/

- Listserv

Weiterführende Links

- Internet Message Access Protocol (IMAP)
  im Internet unter http://www.imap.org/

- Pretty Good Privacy (PGP); Verschlüsselung von Nachrichten
  im Internet unter http://www.gnupg.org/ und http://www.gnupp.de/

2.4 Fileservice

2.4.1 NFS-System einrichten

Szenario

Für dieses Szenario gehe ich davon aus, dass auf dem Rechner vulab1 NetBSD läuft. Der NFS-Server ist bereits eingerichtet, es existiert ein Verzeichnis /daten/nfs das freigeben werden soll. Auf Rechner vulab2 läuft ein beliebiges Unix-System.

Aufgabe

Geben Sie das Verzeichnis /daten/nfs auf dem Rechner vulab1 für alle Rechner im Labornetz per NFS frei (Subnetz 10.1.1.0/255.255.255.0). Binden Sie dieses Verzeichnis danach beim Rechner vulab2 unter dem Pfad /daten/nfs ein. Es soll auch noch einem Neustart von vulab2 noch dort verfügbar sein.

Überprüfung

Mit dem Kommando showmount testen, ob das genannte Verzeichnis vom Server bereitgestellt wird. Versuchen, auf eine Datei unter /daten/nfs zuzugreifen. Über das Kommando df feststellen, ob das genannte Verzeichnis wirklich auf vulab1 liegt. Danach in /etc/fstab bzw. /etc/vfstab nachsehen, ob das Verzeichnis auch bei einem Neustart eingehängt wird.

2.4.2 Samba-System aufsetzen

Über den Samba-Dienst können, ähnlich wie bei NFS für die Unixwelt, Verzeichnisse und Drucker für andere Rechner im Netzwerk als Shares, vor allem für Windowsmaschinen, freigegeben werden. Samba bietet bei der Arbeit allerdings nicht den gleichen Komfort wie NFS, es müssen einige Eigenheiten beachtet werden, z.B.:

Passwortverschlüsselung:
Bei alten Windowsversionen, wie etwa Windows 95 oder Windows NT, wurde das Passwort unverschlüsselt zum Server übertragen. Neuere Microsoft Betriebssysteme authentifizieren sich verschlüsselt. Die Verschlüsselung des Passwortes wird auf dem Server festgelegt. Man muss also darauf achten, dass man entweder einheitlich nur (un-) verschlüsselte Maschinen benutzt, oder dass man alle Clients dazu bringt, das gleiche Authentifizierungsverfahren zu verwenden.

Windows-Gast-Account
Dem unter Windows verfügbaren Gast-Account sollte aus Sicherheitsgründen ein Benutzerprofil mit wenigen Rechten auf dem Unixserver zugeordnet werden. In der Praxis wird dafür fast immer der User nobody verwendet. Diesem Gast-Account kann der Zugriff für jedes angelegte Share einzeln erlaubt oder verweigert werden.

Unix Rechtemaske
Die Rechteverwaltung ist bei Unix- und Windowsmaschinen unterschiedlich. In der Konfigurationsdatei für Samba wird für jedes Share festgelegt, mit welchen Rechten eine neu erstellte Datei im Unixsystem angelegt wird.

Passwortsynchronisation
Unix und Windows verwenden unterschiedliche Codierungsalgorithmen zur Verschlüsselung von Passwörtern. Daher müssen für den Sambadienst die Passwörter, zusätzlich zu den normalen Unixpasswörtern, verwaltet werden.

Szenario

Beide Kursrechner sind mit beliebigen Unixsystemen ausgestattet. Auf vulab1 liegt unter /root/src der zu installierende Samba-Tarball, es existiert ein Verzeichnis /daten/samba.

Aufgabe

Geben Sie das Verzeichnis /daten/samba auf dem Rechner vulab1 für alle Rechner im Labornetz per Samba frei. Der dazu nötige Samba Tarball liegt unter /root/src.

Verwenden Sie für die Konfiguration ihres Systems folgende Parameter:

```
[global]
  workgroup = VULAB
  security = share
  encrypt passwords = yes
  guest account = nobody

[samba]
  comment = Sambaverzeichnis
  path = /daten/samba
  create mask = 0750
  read only = yes
  public = yes
  browseable = yes
```

Binden Sie dieses Verzeichnis danach beim Rechner vulab2 unter dem Pfad /daten/samba ein.

Überprüfung

Versuchen, auf eine Datei unter /daten/samba zuzugreifen. Über das Kommando df feststellen, ob das genannte Verzeichnis wirklich auf vulab1 liegt.

2.4.3 Anonymous FTP mit oftpd

Szenario

Beim Unixrechner vulab1 liegen die Quellen für den FTP-Server im Verzeichnis /root/src. Der Rechner vulab2 kann mit einem beliebigen System vorbereitet werden, es sollte nur ein FTP-Client für Testzwecke vorhanden sein.

Aufgabe

Stellen Sie das Verzeichnis /home/oftpd auf dem Rechner vulab1 per Anonymous FTP zur Verfügung. Installieren Sie dazu den oftp-Daemon aus dem Verzeichnis /root/src.

Überprüfung

Eine Datei ins Verzeichnis /home/oftpd kopieren und testen, ob diese Datei per FTP bezogen werden kann.

2.4.4 Informationsquellen

Verschiedene FTP-Server

- AnomicFTPD; für Windows, Unix und MacOS
  im Internet unter http://www.anomic.de/AnomicFTPServer/

- NcFTPd Server; für Unix und MacOS
  im Internet unter http://www.ncftpd.com/

- oftpd; für Unix
  im Internet unter http://www.time-travellers.org/oftpd/

- pureFTPD; für Windows, Unix und MacOS
  im Internet unter http://www.pureftpd.org/index.shtml/

- WS_FTP Server; für Windows
  im Internet unter http://www.ipswitch.com/downloads/index.html/

- wu-ftpd
  im Internet unter http://www.wu-ftpd.org/

Verschiedene FTP-Clients

- Axy FTP; für Unix
  im Internet unter http://www.wxftp.seul.org/

- BitBeamer; für Windows
  im Internet unter http://www.bitbeamer.com/en/information.html/

- Captain FTP; für MacOS
  im Internet unter http://captainftp.xdsnet.de/

- Fetch; für MacOS
  im Internet unter http://fetchsoftworks.com/

- gFTP; für Unix
  im Internet unter http://gftp.seul.org/

- Interarchie; für MacOS
  im Internet unter http://www.interarchy.com/main/

- kbear; für Unix
  im Internet unter http://sourceforge.net/projects/kbear/

- LeechFTP; für Windows
  im Internet unter http://www.leechftp.de/

- NcFTP Client; für Windows, Unix und MacOS
  im Internet unter http://www.ncftpd.com/

- wget; für Unix
  im Internet unter http://www.gnu.org/software/wget/wget.html/

- WS_FTP; für Windows
  im Internet unter http://www.ipswitch.com/downloads/index.html/

Weiterführende Links

- Andrew File System (AFS); Netzwerk Dateisystem für Unix
  im Internet unter http://www-2.cs.cmu.edu/~AUIS/andrew-home.html/

- Coda; Netzwerk Dateisystem für Windows und Unix
  im Internet unter http://coda.cs.cmu.edu/

- InterMezzo; Netzwerk Dateisystem für Unix
  im Internet unter http://www.inter-mezzo.org/

- Samba; Unix-Implementierung des Server Message Block Protokolls
  im Internet unter http://www.samba.org/

2.5 Security

2.5.1 Netzwerkdienste und Nmap

Da das folgende Szenario in der Rubrik Security aufgeführt ist, sollte man in der Übung eigentlich den Telnet-Server beenden lassen. Im VULab wird allerdings über Telnet und SSH auf die Kursrechner zugegriffen. Deshalb lasse ich den Dienst nicht abstellen.

Szenario

Auf den Kursrechnern können beliebige Betriebssysteme installiert sein, auf beiden steht das Netzwerktool „nmap“ zur Verfügung. Am Rechner vulab2 laufen neben Telnet-, SSH- und FTP-Servern eine Reihe weiterer, beliebiger Netzwerkdienste.

Aufgabe

1. Stellen Sie fest, welche Dienste am Rechner vulab2 laufen.
(Sie können dazu das Netzwerktool nmap benutzen, das sich jeweils auf vulab1 und vulab2 befindet)

2. Deaktivieren Sie auf vulab2 alle Dienste außer Telnet, SSH und FTP.

Überprüfung

Nach Übungsende wird versucht, ob alle Dienste der Szenariodefinition noch erreicht werden können.

2.5.2 Paketfilering mit iptables

Szenario

Am Rechner vulab2 ist ein Unixsystem mit der Firewall-Software iptables installiert. Die Firewall ist so eingerichtet, dass alle geblockten Pakete in der Datei /var/log/messages protokolliert werden. Außerdem läuft ein Webserver am Standardport 80. Der Rechner vulab1 wird zur eigentlichen Übung nicht benötigt. Er dient nur dazu, um den Erfolg der Firewall-Konfiguration zu testen. Auf dem Rechner vulab1 kann ein beliebiges Betriebssystem installiert sein.

Aufgabe

Konfigurieren Sie die iptables-Firewall am Rechner vulab2 so, dass er keine Anfragen auf den Webserver von vulab2 vom Rechner vulab1 aus annimmt.

Für alle anderen Rechner soll der Webserver weiterhin erreichbar sein.

Überprüfung

Man versucht von allen Rechnern im VULab aus, auf den Webserver auf vulab2 zuzugreifen. Wenn nur vulab1 den Webserver nicht ansprechen kann, sollte die Firewall richtig arbeiten. Am Ende überprüft man anhand der Logdatei, ob die gewünschten Pakete wirklich von der Firewall geblockt wurden. Sind darin entsprechende Einträge verzeichnet und ist der Zeitpunkt der Änderung der Logdatei sinnvoll gesetzt?

2.5.3 Informationsquellen

Verschiedene Portscan-Tools

- Nmap; für Windows, Unix und MacOS
  im Internet unter http://www.insecure.org/nmap/

- PortScan; für Windows
  im Internet unter http://www.shareup.com/PortScan-download-9904.html/

Verschiedene Personal Firewall Software

- Kerio Personal Firewall; für Windows
  im Internet unter http://www.kerio.com/kpf_home.html/

- McAfee; für Windows
  im Internet unter http://us.mcafee.com/default.asp/

- Norton Personal Firewall; für Windows
  im Internet unter http://www.symantec.com/

- Outpost Firewall; für Windows
  im Internet unter http://www.agnitum.com/

- Tiny Firewall; für Windows
  im Internet unter http://www.tinysoftware.com/home/tiny2?la=DE/

- Zone Alarm; für Windows
  im Internet unter http://www.zonelabs.com/

Verschiedene Firewall (DMZ) Software

- IP Filter; für Unix
  im Internet unter http://coombs.anu.edu.au/~avalon/

- Kerio WinRoute Firewall; für Windows
  im Internet unter http://www.kerio.com/kwf_home.html/

- netfilter/iptales; für Unix
  im Internet unter http://www.netfilter.org/

2.6 Newssever

2.6.1 Leafnode Newsserver installieren

Szenario

Am Unix-Rechner vulab1 liegen im Verzeichnis /root/src die Sourcen zu einem Leafnode Newsserver.

Aufgabe

1. Installieren Sie auf dem Rechner vulab1 einen lokalen Newsserver. Verwenden Sie dazu die Leafnode-Sourcen aus dem Verzeichnis /root/src.

2. Konfigurieren Sie den Server so, dass
   - er die Newsgruppen vom Server news.freenet.de bezieht.
   - er nur die Beiträge bezieht, die nicht älter als 30 Tage sind.
   - er nicht gelesene Beiträge nach 15 Tagen löscht.
   - er täglich nach neuen Artikeln auf dem externen Server sucht.

Überprüfung

1. Testen, ob der Server läuft.

2. Testen, ob die Liste aller vorhandenen Newsgruppen in der Datei /var/spool/news/leaf.node/groupinfo angelegt wurde.

3. Prüfen, ob Cronjobs zum Löschen und Abholen von Nachrichten erzeugt wurden.

2.6.2 Informationsquellen

Verschiedene Newsserver

- Hamster; für Windows
  im Internet unter http://www.tglsoft.de/

- INN; für Unix
  im Internet unter http://www.isc.org/

- Leafnode; für Windows und Unix
  im Internet unter http://leafnode.sourceforge.net/ und http://www.leafnode.org/

- Noffle; für Unix
  im Internet unter http://noffle.sourceforge.net/

Verschiedene Newsreader

- Gnus; für Unix
  im Internet unter http://www.gnus.org/

- KNode; für Unix
  im Internet unter http://knode.sourceforge.net/

- Mozilla; für Windows, Unix und MacOS
  im Internet unter http://www.mozilla.org/

- MS Outlook Express; für Windows und MacOS
  im Internet unter http://www.microsoft.com/

- Pine; für Windows und Unix
  im Internet unter http://www.washington.edu/pine/

- Slnr; für Windows, Unix und MacOS
  im Internet unter http://www.slrn.org/

- Sylpheed; für Unix
  im Internet unter http://sylpheed.good-day.net/index.cgi.en/

- tin; für Unix
  im Internet unter http://www.tin.org/

- Xnews; für Windows
  im Internet unter http://xnews.newsguy.com/

2.7 Shellprogrammierung

Im Bereich Shellprogrammierung sind die unterschiedlichsten Problemstellungen denkbar, die die Erstellung von und den Umgang mit Shellscripten zur Routine machen sollen. Eigentlich ist egal, welches Ergebnis das Script liefert. Das Hauptziel sollte sein, mit möglichst vielen unterschiedlichen Shellstrukturen zu arbeiten, um diese kennen zu lernen.

Ich habe mich bemüht, vielleicht doch eine Aufgabenstellung zu finden, die so auch später im Leben eines Systemverwalters auftreten kann. Generell kann dieser Übungsaufbau aber für beliebige andere Aufgabendefinitionen verwendet werden.

Ich fordere in meinem Szenario die Erstellung von Bourne Shell Scripten, da diese voraussichtlich auf allen gängigen Unix-Systemen lauffähig sein sollten. Ähnliche Aufgaben wären auch für Perl Scripten möglich, da diese ebenfalls auf vielen Systemen – z.B. Windows – einsetzbar sind.

Szenario

Beim Kursrechner vulab1 ist ein Unixsystem installiert, natürlich mit einem Bourne Shell Interpreter. Im Verzeichnis /home/dummy liegen eine Reihe beliebiger Verzeichnisse und Dateien, die eine Standardumgebung für einen Account darstellen sollen. Im Verzeichnis /home liegt eine Datei newusers mit mehreren anzulegenden Benutzeraccounts. Diese Datei hat die Form:

Login1:Passwort1
Login2:Passwort2
…

Aufgabe

1. Erstellen Sie am Rechner vulab1 im Verzeichnis /root/sysadmin ein Bourn Shell Script mit dem Namen myadduser das folgendes leistet:

- Das Script soll mehrere Benutzeraccounts anlegen. Die Loginnamen und die zugehörigen Passwörter sollen aus einer Datei, die über die Kommandozeile als Parameter übergeben wird, entnommen werden.

- In die Homeverzeichnisse der neuen Accounts soll der Inhalt des Verzeichnisses /home/dummy kopiert werden.

2. Testen Sie Ihr Script mit der Datei unter /home/newusers, in der mehrere Loginnamen mit Passwort neuer Accounts angegeben sind.

Überprüfung

1. Das Script mit einer anderen Benutzerquelldatei ausführen.

2.Testen, ob alle neuen Benutzer aus dieser Datei richtig angelegt wurden, und ob die Homeverzeichnisse wirklich den jeweiligen Nutzern gehören.

3. Überprüfen, ob die gewünschten Dateien ins Homeverzeichnis kopiert wurden, und ob auch diese dem jeweiligen Benutzer gehören.

Schlusswort

Eine erschöpfende Aus- und Weiterbildung für die einzelnen Themenbereiche ist wohl während der kurzen Bearbeitungsdauer einer VULab-Übung nicht möglich. Dafür wurde das VULab allerdings auch nicht ins Leben gerufen. Vielmehr sollen die Studenten den Umgang mit Unixmaschinen üben, Informationen über den Aufbau eines solchen Systems gewinnen und so sicherer in der Bedienung werden. Darüber hinaus wird den Studenten die Möglichkeit geben in die Konfiguration und den Aufbau grundlegender Netzwerktechniken hineinzuschnuppern.

Bei der endgültigen Übungszusammenstellung solle man darauf achten, dass unterschiedliche Techniken zur Systemadministration und –installation angewendet werden. Jede Software unter Windows kann durch das Ausführen einer .exe-Datei aufgesetzt werden. So können Server ohne die geringste Ahnung vom Betriebssystem oder dem Dienst selbst betrieben werden. Ähnlich verhält es sich unter Unix bei einer Installation durch ./configure – make – make install. Deshalb sollten im Labor Aufgaben einmal über die Compilierung des Quellcodes, ein anderes Mal über Paket-Management-Systeme, wieder ein anderes Mal über die Installation eines .tar-Archives geschehen. Eine weitere Variationsmöglichkeit ist, ein vollständig installiertes System vorzugeben, bei dem dann nach Fehlern in der Konfiguration gesucht werden soll.

Da ich nur Themenvorschläge liefere und nicht weiß, welche meiner Übungen letztlich umgesetzt werden, habe ich die einzelnen Szenarien nicht aufeinander abgestimmt.

Manche der aufgeführten Themen sind, so denke ich, ohne Vorkenntnisse schwer zu bearbeiten. Es kann trotzdem sinnvoll sein Aufgaben dazu zu formulieren, z.B. wenn

- Grundlagen dazu vorher im Rahmen einer Vorlesung übermittelt werden.
- in der Aufgabenstellung eine Einführung zum Thema und/oder weiterführende Links mit angegeben sind.
- sich der Student selbst die nötigen Informationen beschaffen und mit dem Thema befassen soll. Diese Arbeit, also das eigenständige Einarbeiten in eine Thematik, ist bei der Systempflege nicht untypisch, kann also durchaus Bestandteil im VULab sein.

Auch solche Vorbedingungen sind mir nicht bekannt und ich habe deshalb auch darauf bei der Erstellung von Übungsvorlagen keine Rücksicht genommen.

Anhang A
Quellcode

Listing A.1: adressen.ldif

```
dn: o=intern
objectclass: organization
o: intern

dn: cn=admin, o=intern
objectclass: person
cn: admin
description: LDAP-Administrator

dn: cn=vulabPerson, o=intern
objectclass: person
cn: vulabPerson
telephonenumber: 151
mail: vulabPerson@intern
description: Person fuer das vulab
```

Listing A.2: check-file-contains

```
#!/usr/bin/perl
##############################################################################
####
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Tested ob $FILENAME den Text $CONTENT_SHOULD
***
';

##############################################################################
####
# Parameter:
# [ "Variable", "Default, "Beschreibung der Variable" ]
#
@vars = (
          [ "FILENAME", "",
            "zu durchsuchendes File; absoluter Pfad" ],
          [ "CONTENT_SHOULD", "",
            "zu suchender Text/Regulaerer Ausdruck" ]
        );

```

```
22 ##########################################################################
   ####
23 # Check-Spezifisch:
24 sub check()
25 {
26    $temp =  `cat $FILENAME | egrep -e "$CONTENT_SHOULD" | wc -l`;
27
28    print "FILENAME=$FILENAME\n";
29    print "CONTENT_SHOULD=$CONTENT_SHOULD\n";
30    print "";
31
32    if ( $temp >= 1 ) {
33       $rc="ok";
34    } else {
35       $rc="wrong";
36    }
37
38    print "$rc\n";
39 }
40
41 ##########################################################################
   ####
42 ### Common code:
43
44 # Variablen übernehmen
45 sub init()
46 {
47    for($i=0; $i<=$#vars; $i++) {
48       if(exists($ENV{$vars[$i][0]})) {
49          ${$vars[$i][0]} = $ENV{$vars[$i][0]};
50       } else {
51          ${$vars[$i][0]} = $vars[$i][1];
52       }
53    }
54 }
55
56 # "Hauptprogramm"
57 if($ARGV[0] eq "listparms") {
58    for($i=0;$i<=$#vars;$i++) {
59       print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
60    }
61 } elsif($ARGV[0] eq "whatis") {
62    $WHATIS=~s/^\n*//g;
63    $WHATIS=~s/\n\*\*\* ?//g;
64    $WHATIS=~s/^\*\*\* ?//g;
65    $WHATIS=~s/\n*$//g;
66    print "$WHATIS\n";
67 } elsif($ARGV[0] eq "-h") {
68    print "whatis     Kurzbeschreibung des Scripts\n";
69    print "listparms  Listet Variablen mit Default und Beschreibung\n";
70    print "-h         Alle Parameter\n";
71    print "sonst      Check-Script wird ausgefuehrt\n";
72 } else {
73    init();
74    check();
75 }
```

Listing A.3: check-file-exists

```
1  #!/usr/bin/perl
2  ##############################################################################
   ####
3  # Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
4  $WHATIS='
5  ***
6  *** Tested ob die Datei $FILENAME existiert
7  ***
8  ';
9
10 ##############################################################################
   ####
11 # Parameter:
12 # [ "Variable", "Default, "Beschreibung der Variable" ]
13 #
14 @vars = (
15          [ "FILENAME", "",
16            "gesuchtes File; absoluter Pfad" ]
17        );
18
19 ##############################################################################
   ####
20 # Check-Spezifisch:
21 sub check()
22 {
23    $temp = `test -f $FILENAME`;
24
25    print "FILENAME=$FILENAME\n";
26    print "";
27
28    if ( $? == 0 ) {
29       $rc="ok";
30    } else {
31       $rc="wrong";
32    }
33
34    print "$rc\n";
35 }
36
37 ##############################################################################
   ####
38 ### Common code:
39
40 # Variablen übernehmen
41 sub init()
42 {
43    for($i=0; $i<=$#vars; $i++) {
44       if(exists($ENV{$vars[$i][0]})) {
45          ${$vars[$i][0]} = $ENV{$vars[$i][0]};
46       } else {
47          ${$vars[$i][0]} = $vars[$i][1];
48       }
49    }
50 }
51
52 # "Hauptprogramm"
53 if($ARGV[0] eq "listparms") {
54    for($i=0;$i<=$#vars;$i++) {
55       print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
56    }
57
```

```
58 } elsif($ARGV[0] eq "whatis") {
59    $WHATIS=~s/^\n*//g;
60    $WHATIS=~s/\n\*\*\* ?//g;
61    $WHATIS=~s/^\*\*\* ?//g;
62    $WHATIS=~s/\n*$//g;
63    print "$WHATIS\n";
64 } elsif($ARGV[0] eq "-h") {
65    print "whatis     Kurzbeschreibung des Scripts\n";
66    print "listparms  Listet Variablen mit Default und Beschreibung\n";
67    print "-h         Alle Parameter\n";
68    print "sonst      Check-Script wird ausgefuehrt\n";
69 } else {
70    init();
71    check();
72 }
```

Listing A.4: check-finger

```
 1 #!/usr/bin/perl
 2 ##############################################################################
   ####
 3 # Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
 4 $WHATIS='
 5 ***
 6 *** Tested per finger, ob der Account $USER_SHOULD bekannt ist
 7 ***
 8 ';
 9
10 ##############################################################################
   ####
11 # Parameter:
12 # [ "Variable", "Default, "Beschreibung der Variable" ]
13 #
14 @vars = (
15          [ "USER_SHOULD", "vulab",
16            "gesuchter Account" ]
17        );
18
19 ##############################################################################
   ####
20 # Check-Spezifisch:
21 sub check()
22 {
23    $temp = `finger $USER_SHOULD | wc -l`;
24
25    print "finger $USER_SHOULD?\n";
26    print "";
27
28    if ( $temp >0 ) {
29       $rc="ok";
30    } else {
31       $rc="wrong";
32    }
33    print "$rc\n";
34 }
35
36 ##############################################################################
   ####
37 ### Common code:
```

```
38
39 # Variablen übernehmen
40 sub init()
41 {
42    for($i=0; $i<=$#vars; $i++) {
43       if(exists($ENV{$vars[$i][0]})) {
44          ${$vars[$i][0]} = $ENV{$vars[$i][0]};
45       } else {
46          ${$vars[$i][0]} = $vars[$i][1];
47       }
48    }
49 }
50
51 # "Hauptprogramm"
52 if($ARGV[0] eq "listparms") {
53    for($i=0;$i<=$#vars;$i++) {
54       print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
55    }
56 } elsif($ARGV[0] eq "whatis") {
57    $WHATIS=~s/^\n*//g;
58    $WHATIS=~s/\n\*\*\* ?//g;
59    $WHATIS=~s/^\*\*\* ?//g;
60    $WHATIS=~s/\n*$//g;
61    print "$WHATIS\n";
62 } elsif($ARGV[0] eq "-h") {
63    print "whatis     Kurzbeschreibung des Scripts\n";
64    print "listparms  Listet Variablen mit Default und Beschreibung\n";
65    print "-h         Alle Parameter\n";
66    print "sonst      Check-Script wird ausgefuehrt\n";
67 } else {
68    init();
69    check();
70 }
```

Listing A.5: check-program-output

```
 1 #!/usr/bin/perl
 2 ##############################################################################
   ####
 3 # Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
 4 $WHATIS='
 5 ***
 6 *** Tested ob ein Programm die richtige Antwort zurückliefert
 7 ***
 8 ';
 9
10 ##############################################################################
   ####
11 # Parameter:
12 # [ "Variable", "Default, "Beschreibung der Variable" ]
13 #
14 @vars = (
15          [ "PROGRAM", "domainname",
16            "auszufuehrendes Programm" ],
17          [ "OUTPUT_SHOULD", "fehler",
18            "erwartete Ausgabe" ]
19        );
20
21 ##############################################################################
```

```
####
# Check-Spezifisch:
sub check()
{
   $output_is = `$PROGRAM`;

   print "PROGRAM=$PROGRAM\n";
   print "output_should=$OUTPUT_SHOULD\n";
   print "output_is=$output_is\n";
   print "";

   if ( $output_is = $OUTPUT_SHOULD ) {
      $rc="ok";
   } else {
      $rc="wrong";
   }

   print "$rc\n";
}

##############################################################################
####
### Common code:

# Variablen übernehmen
sub init()
{
   for($i=0; $i<=$#vars; $i++) {
      if(exists($ENV{$vars[$i][0]})) {
         ${$vars[$i][0]} = $ENV{$vars[$i][0]};
      } else {
         ${$vars[$i][0]} = $vars[$i][1];
      }
   }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
   for($i=0;$i<=$#vars;$i++) {
      print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
   }
} elsif($ARGV[0] eq "whatis") {
   $WHATIS=~s/^\n*//g;
   $WHATIS=~s/\n\*\*\* ?//g;
   $WHATIS=~s/^\*\*\* ?//g;
   $WHATIS=~s/\n*$//g;
   print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
   print "whatis     Kurzbeschreibung des Scripts\n";
   print "listparms  Listet Variablen mit Default und Beschreibung\n";
   print "-h         Alle Parameter\n";
   print "sonst      Check-Script wird ausgefuehrt\n";
} else {
   init();
   check();
}
```

Listing A.6: check-rpc-reg

```
1  #!/usr/bin/perl
2  ##########################################################################
   #####
3  # Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
4  $WHATIS='
5  ***
6  *** Tested ob der RPC-Dienst $PROGRAM_NR registriert wurde. Zoordnung
   Programmnummer/Dienst unter /etc/rpc
7  ***
8  ';
9
10 ##########################################################################
   ####
11 # Parameter:
12 # [ "Variable", "Default, "Beschreibung der Variable" ]
13 #
14 @vars = (
15          [ "PROGRAM_NR", "0815",
16            "Programmnummer des Dienstes der ueberprueft werden soll" ]
17        );
18
19 ##########################################################################
   ####
20 # Check-Spezifisch:
21 sub check()
22 {
23    $temp =  `rpcinfo -p | grep $PROGRAM_NR | wc -l`;
24
25    print "PROGRAM_NR=$PROGRAM_NR running?\n";
26    print "";
27
28    if ( $temp >= 1 ) {
29       $rc="ok";
30    } else {
31       $rc="wrong";
32    }
33
34    print "$rc\n";
35 }
36
37 ##########################################################################
   ####
38 ### Common code:
39
40 # Variablen übernehmen
41 sub init()
42 {
43    for($i=0; $i<=$#vars; $i++) {
44       if(exists($ENV{$vars[$i][0]})) {
45          ${$vars[$i][0]} = $ENV{$vars[$i][0]};
46       } else {
47          ${$vars[$i][0]} = $vars[$i][1];
48       }
49    }
50 }
51
52 # "Hauptprogramm"
53 if($ARGV[0] eq "listparms") {
54    for($i=0;$i<=$#vars;$i++) {
55       print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
```

```
   }
} elsif($ARGV[0] eq "whatis") {
   $WHATIS=~s/^\n*//g;
   $WHATIS=~s/\n\*\*\* ?//g;
   $WHATIS=~s/^\*\*\* ?//g;
   $WHATIS=~s/\n*$//g;
   print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
   print "whatis     Kurzbeschreibung des Scripts\n";
   print "listparms  Listet Variablen mit Default und Beschreibung\n";
   print "-h         Alle Parameter\n";
   print "sonst      Check-Script wird ausgefuehrt\n";
} else {
   init();
   check();
}
```

Listing A.7: nis-check-ypcat

```
#!/usr/bin/perl
################################################################################
# Kurzbeschreibung: "Dieses Script ($0) $WHATIS\n." (1 Zeile)
$WHATIS='
***
*** Tested ob der Account $USER_SHOULD in ypcat passwd bekannt ist
***
';

################################################################################
# Parameter:
# [ "Variable", "Default, "Beschreibung der Variable" ]
#
@vars = (
        [ "USER_SHOULD", "vulab",
          "gesuchter Account" ]
       );

################################################################################
# Check-Spezifisch:
sub check()
{
   $temp = `ypcat passwd | grep $USER_SHOULD | wc -l`;

   print "ypcat passwd $USER_SHOULD?\n";
   print "";

   if ( $temp >0 ) {
      $rc="ok";
   } else {
      $rc="wrong";
   }

   print "$rc\n";
}

################################################################################
```

```
####
### Common code:

# Variablen übernehmen
sub init()
{
   for($i=0; $i<=$#vars; $i++) {
      if(exists($ENV{$vars[$i][0]})) {
         ${$vars[$i][0]} = $ENV{$vars[$i][0]};
      } else {
         ${$vars[$i][0]} = $vars[$i][1];
      }
   }
}

# "Hauptprogramm"
if($ARGV[0] eq "listparms") {
   for($i=0;$i<=$#vars;$i++) {
      print "$vars[$i][0]|$vars[$i][1]|$vars[$i][2]\n";
   }
} elsif($ARGV[0] eq "whatis") {
   $WHATIS=~s/^\n*//g;
   $WHATIS=~s/\n\*\*\* ?//g;
   $WHATIS=~s/^\*\*\* ?//g;
   $WHATIS=~s/\n*$//g;
   print "$WHATIS\n";
} elsif($ARGV[0] eq "-h") {
   print "whatis     Kurzbeschreibung des Scripts\n";
   print "listparms  Listet Variablen mit Default und Beschreibung\n";
   print "-h         Alle Parameter\n";
   print "sonst      Check-Script wird ausgefuehrt\n";
} else {
   init();
   check();
}
```

Listing A.8: passwd

```
#!/usr/bin/perl

########################## Parameter ##########################

#1 Kommando fuer aktuellen Benutzer anwenden, falls kein Parameter
angegeben wird. Noetig fuer #9
if(!exists($ARGV[0]))
{
   $ARGV[0]=$ENV{USER};
}

#2 auszufuehrendes Script
$script="/usr/bin/dwssap";

#3 Passwort-Abfrage ueber
$method="ypcat passwd";

#4 Trennzeichen bei der Suche nach dem Passwort
$separator=":";

#5 Passwort ist im Feld Nr:
```

```
21 $index=1;
22
23 #6 Log-Eintraege fuer den Account
24 $account="test";
25
26 #7 gesuchtes Passwort fuer den Account
27 $password="neues";
28
29 #8 Log-Ausgabe in die Datei
30 $file="/var/log/passwd.log";
31
32 ########################## Programm ##########################
33
34 #9 Hier wird ein Wert fuer ARGV[0] benoetigt, sonst faellts auf!
35 print "Changing password for $ARGV[0].\n";
36
37 $startproc=`$script $ARGV[0]`;
38
39 #10 Ausgabe des Rueckgabewertes nach der Passwortaenderung
40 @temp_rc = split("\n",$startproc);
41 if(exists($temp_rc[1]))
42 {
43    print $temp_rc[1]."\n";
44 }
45
46 if($ARGV[0] eq $account)
47 {
48    #11 Suche nach dem Account
49    $pw_current=`$method | grep $account`;
50
51    #12 Passwort isolieren
52    @temp = split($separator,$pw_current);
53
54    $salt=substr(@temp[$index],0,2);
55
56    $pw_crypted=crypt($password,$salt);
57
58    open(FILE, ">> $file");
59
60    if(@temp[$index] eq $pw_crypted)
61    {
62       print FILE "passwd: $account->$password; ok\n";
63    }
64    else
65    {
66       print FILE "passwd: $account->$password; wrong\n";
67    }
68
69    close(FILE);
70 }
```

Listing A.9: sldapd.conf

```
1 # LDAP-Server
2 # /etc/openldap/sldapd.conf
3
4 # Schemacheck ausschalten
5 schemacheck off
6
7 # Die Datenbank hat das Format ldbm und befindet sich im Verzeichnis
```

```
   /var/lib/ldap.
 8 database ldbm
 9 directory /var/lib/ldap
10
11 # Die Zugriffsattribute sollen nicht erstellt werden.
12 lastmod off
13
14 # Die Datenbank fuehlt sich fuer Anfragen zustaendig, die unterhalb von
   o=intern liegen.
15 suffix "o=intern"
16
17 # Der Administrator ist cn=admin, o=intern mit dem Passwort vulab
18 rootdn "cn=admin, o=intern"
19 rootpw vulab
20
21 # Das Defaultrecht ist lesen. Auf moegliche Attribute userPassword hat
   nur der Eigentuemer schreibende Rechte. Allen anderen ist der Zugriff
   untersagt. Auch das Administratorobjekt ist fuer jedermann unsichtbar.
22 defaultaccess read
23
24 access to attr=userPassword
25        by self write
26        by * none
27
28 access to dn="cn=admin,o=intern"
29        by * none
30
31 access to *
32        by * read
```

Anhang B
Beiliegende CD

Abbildungsverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

Hubert Feyrer
Systemverwaltung unter Unix – Sommersemester 2004
http://rfhs8012.fh-regensburg.de/~feyrer/SA/SS2004/, 2004

Hubert Feyrer
Virtuelles Unix Labor
http://www.feyrer.de/vulab/, 2004

Jens Banning
Linux Netzwerkadministration – Installation und Konfiguration von Netzwerkdiensten
Addison-Wesley Verlag, 2002

Jörg Holzmann, Jürgen Plate
Linux-Server für Intranet und Internet – Den Server einrichten und administrieren
Carl Hanser Verlag, 2002

Michael Kofler
Linux – Installation, Konfiguration, Anwendung
Addison-Wesley Verlag, 2001

Rosa Riebl
NetBSD 1.6 – Installieren, Konfigurieren, Administrieren
C&L Computer und Literaturverlag, 2003