



THE SOUND OF SUNSHINE

Group 3

Adrian-Jorge Sanchez (adrianjorge.sanchez)

Index

1. Summary of the Practice	3
2. Files Modified/Generated.....	4
3. Scripts	5
Index.sh.....	5
Menu.sh.....	7
Process_management.sh.....	8
Monitoring.sh	9
Shutdown_restart.sh	10
Manage_logs.sh.....	11
User_management.sh	12
Packet_filtering.sh	13
Manage_tasks.sh	14
Music_search.sh	15
4. Observed Problems and Conclusion	16

1. Summary of the Practice

After successfully installing Linux From Scratch (LFS), the next step involved assigning a purpose to the system by implementing a remote management web service.

The objective was to configure LFS properly and create a web server with functionalities accessible through a web interface using shell scripts and HTML code.

Key functionalities to be implemented were login, process management, monitoring, shutdown and restart, log management, user management, packet filtering, managw preprogrammed tasks and music search.

This comprehensive set of functionalities aimed to provide remote control and management of the LFS system through a web interface, enhancing user experience and system administration capabilities.

2. Files Modified/Generated

With the objective of successfully implementing this practice, we installed `apache-2.4.58` to create the web server. We followed the BLFS installation guide of `apache` and once finished, we created a folder in `/srv/www/` called `ASO` in order to allocate all of the scripts.

Furthermore, we created one script per page and gave the `apache` user permissions to execute those scripts. Additionally, in the `/etc/httpd/httpd.conf` the following changes were made:

```
<IfModule !mpm_prefork_module>
    LoadModule cgid_module /usr/lib/httpd/modules/mod_cgid.so
</IfModule>
<IfModule mpm_prefork_module>
    LoadModule cgi_module /usr/lib/httpd/modules/mod_cgi.so
</IfModule>
```

The lines `LoadModule` were commented however, they should be uncommented. Thus, the configurations set in each `if` are applied depending on the case.

```
ServerName 192.186.131
```

`ServerName` line was added with the IP of the machine. Hence, when the user enters, the IP goes to the webserver.

```
DocumentRoot "/srv/www/aso"
<Directory "/srv/www/aso">
    #
    # Possible values for the Options directive are
    # or any combination of:
    #   Indexes Includes FollowSymLinks SymLinks
    #
    # Note that "MultiViews" must be named *explicitly*
    # doesn't give it to you.
    #
    # The Options directive is both complicated and
    # http://httpd.apache.org/docs/2.4/mod/core.html#options
    # for more information.
    #
    #Options Indexes FollowSymLinks
    DirectoryIndex index.sh
    Options +Indexes +FollowSymLinks +ExecCGI
    AddHandler cgi-script .cgi .sh
```

Lastly, the path from previously mentioned folder `ASO`, is now set as the document root instead of `<Directory "/srv/www/">`. Moreover, inside, the `DirectoryIndex` line was added in order to automatically redirect them to the `index.sh` once the IP is entered. The `Options` line enables different features, such as, being able to execute `CGI` scripts or navigate from one file to another (switching scripts). Finally, the `AddHandler` line tells `apache` to treat `.cgi` and `.sh` files as `CGI` scripts.

3. Scripts

Index.sh

The following script shows our login page, containing a field for the username, another one for the password and a button to log in with all the data entered.

Once the button is pressed, it submits the form as a post request to `index.sh`. Consequently, `index.sh` is executed again and when it checks the request method, as it is a post, it gets the data passed, which is the username and password. Moreover, once it gets both, it checks the `/etc/passwd` for the username entered. When found, it gets the hashed password and the salt from the user (from the `/etc/shadow`) and generates the hashes (entered password with the salt got) and compares it to the password from the shadow file. If they match, it will go to the script `menu.sh` and pass the argument. However, if they are not the same, it will go to the `index.sh` script (login).

```
#!/bin/bash
if [ "$REQUEST_METHOD" = "POST" ]; then
    read -r POST_DATA

    username=$(echo "$POST_DATA" | grep -o 'username=[^&]*' | cut -d= -f2)
    password=$(echo "$POST_DATA" | grep -o 'password=[^&]*' | cut -d= -f2)

    if grep -q "^$username:" /etc/passwd; then
        hashed_password=$(sudo grep "^$username:" /etc/shadow | cut -d':' -f2)
        salt=$(sudo grep "^$username:" /etc/shadow | cut -d'$' -f3)
        entered_hashed_password=$(sudo openssl passwd -1 -salt "$salt" "$password")

        if [ "$hashed_password" == "$entered_hashed_password" ]; then
            echo "Location: menu.sh?username=$username"
            #echo "$(date '+%Y-%m-%d %H:%M:%S') User '$username' successfully logged in" >> "/var/log/httpd/actions.log"
            logger -p user.info "'$username' successfully logged in"
        else
            echo "Location: index.sh"
            #echo "$(date '+%Y-%m-%d %H:%M:%S') User '$username' failed to log in" >> "/var/log/httpd/actions.log"
            logger -p user.info "'$username' failed to log in"
        fi
    else
        logger -p user.info "'$username' failed to log in"
        echo "Location: index.sh "
    fi
    echo ""
fi
```

For this script and the following ones, the `sudo` command has been installed following the guide from the BLFS book. Additionally, in the file `/etc/sudoers`, privileges are assigned to the `apache` user.

```
##
## User privilege specification
##
root ALL=(ALL:ALL) ALL
apache ALL=(ALL:ALL) NOPASSWD: ALL
```

Lastly, the hashing method must be checked by looking at the `/etc/shadow` file, in our case it was the `yescrypt` type. As it cannot be generated with any of the installed tools, we changed it to the `MD5` type. By changing the encryption method in the `/etc/login.defs` file.

```
# THE PAM MODULES CONTROL  
#  
ENCRYPT_METHOD MD5
```

Menu.sh

Menu.sh script was used to display all of the different options available. Depending on the selection, it will redirect them to the appropriate script. This is achieved by sending a post request to itself with the value of the button pressed. Moreover, if it is a post request it gets the data passed and based on the value it passes the username and changes to the corresponding page.

```
#!/bin/bash
username=$(echo "$QUERY_STRING" | grep -o 'username=[^&]*' | cut -d= -f2)

if [ "$REQUEST_METHOD" == "POST" ]; then
    read -n $CONTENT_LENGTH post_data
    selected_option=$(echo "$post_data" | cut -d'=' -f2)

    case $selected_option in
        1)
            echo "Location: process_management.sh?username=$username"
            logger -p user.info "'$username' went to the process management page"
            ;;
        2)
            echo "Location: monitoring.sh?username=$username"
            logger -p user.info "'$username' went to the monitoring page"
            ;;
        3)
            echo "Location: shutdown_restart.sh?username=$username"
            logger -p user.info "'$username' went to the shutdown & restart page"
            ;;
        4)
            echo "Location: manage_logs.sh?username=$username"
            logger -p user.info "'$username' went to the logs page"
            ;;
        5)
            echo "Location: user_management.sh?username=$username"
            logger -p user.info "'$username' went to the management page"
            ;;
        6)
            echo "Location: packet_filtering.sh?username=$username"
            logger -p user.info "'$username' went to the packet filtering page"
            ;;
        7)
            echo "Location: manage_tasks.sh?username=$username"
            logger -p user.info "'$username' went to the task management page"
            ;;
        8)
            echo "Location: music_search.sh?username=$username"
            logger -p user.info "'$username' went to the music search page"
            ;;
    esac
fi
```

Process_management.sh

This script displays a list of existing processes with an empty field in order to enter the PID of a process. Moreover, we also added a drop-down menu to choose the action to perform and a button to submit the entered data through a post request.

- If the script is called through a post request, it gets the data and checks the type of action.
- If it is status, it gets the status of the process matching the PID entered and displays it.
- If it is interrupt, it stops the process matching the PID entered during the amount of time entered.
- Finally, if it is remove, it kills the process matching the PID entered.

```
if [ "$REQUEST_METHOD" = "POST" ]; then
    read -r QUERY_STRING
    IFS='&' read -ra params <<< "$QUERY_STRING"
    for param in "${params[@]"; do
        IFS='=' read -r key value <<< "$param"
        case "$key" in
            pid) pid_param="$value" ;;
            action) action_param="$value" ;;
            seconds) seconds_param="$value" ;;
        esac
    done

    if [ "$pid_param" == "back" ]; then
        echo "Location: menu.sh?username=$username"
        #echo "$(date +%Y-%m-%d %H:%M:%S) User went back to main menu" >> "/var/log/httpd/actions.log"
        logger -p user.info "'$username' went back to main menu"
    fi

    if [ "$action_param" == "status" ]; then
        get_process_status "$pid_param"
        #echo "$(date +%Y-%m-%d %H:%M:%S) User got the state of the process with PID: $pid_param" >> "/var/log/httpd/actions.log"
        logger -p user.info "'$username' got the state of the process with PID: $pid_param"
    fi

    if [ "$action_param" == "interrupt" ]; then
        interrupt_process "$pid_param" "$seconds_param"
        #echo "$(date +%Y-%m-%d %H:%M:%S) User interrupted during $seconds_param seconds the process with PID: $pid_param" >> "/var/log/httpd/actions.log"
        logger -p user.info "'$username' interrupted during $seconds_param seconds the process with PID: $pid_param"
    fi

    if [ "$action_param" == "remove" ]; then
        remove_process "$pid_param"
        #echo "$(date +%Y-%m-%d %H:%M:%S) User removed process with PID: $pid_param" >> "/var/log/httpd/actions.log"
        logger -p user.info "'$username' removed process with PID: $pid_param"
    fi
fi
```


Monitoring.sh

The Monitoring.sh displays three sections:

1. Resources of the machine.
2. Last 10 accesses to the server.
3. Basic information of the system.

For the system resources, the `uptime` command is used to get the time of the system, load average and users logged in. The `free -h` is used to display the information regarding the system's memory usage in human-readable format. Finally, `df -h` is used to display the information about the disk space usage in human-readable format.

For section 2, the last 10 lines of the file `/var/log/httpd/actions.log` where the word `log` appears were displayed. `Actions.log` is the file where each action of each user perform is being logged.

Lastly, the additional information is achieved with the use of `uname -a` command to display information such as the kernel name and version, hostname, operating system...

```
<pre>$ (uptime)</pre>
<pre>$ (free -h)</pre>
<pre>$ (df -h)</pre>
<h2>Last 10 Server Accesses:</h2>
<pre>$ (grep 'log' /var/log/httpd/actions.log | tail -n 10)</pre>
<h2>System information:</h2>
<pre>$ (uname -a)</pre>
```

Shutdown_restart.sh

Shutdown_restart.sh displays 2 buttons in order to allow the user to restart or shutdown (1 option per button). Based on the one clicked it will either use /etc/rc.d/init.d/httpd stop to shutdown the web server or /etc/rc.d/init.d/httpd restart to restart it.

```
if [ "$REQUEST_METHOD" = "POST" ]; then
    read -r QUERY_STRING
    IFS='&' read -ra params <<< "$QUERY_STRING"
    for param in "${params[@]}; do
        IFS='=' read -r key value <<< "$param"
        case "$key" in
            action)
                case "$value" in
                    shutdown)
                        /etc/rc.d/init.d/httpd stop
                        #echo "$(date '+%Y-%m-%d %H:%M:%S') User shutdowned the server" >> "/var/log/httpd/actions.log"
                        logger -p user.info "'$username' shutdowned the server"
                        ;;
                    restart)
                        /etc/rc.d/init.d/httpd restart
                        #echo "$(date '+%Y-%m-%d %H:%M:%S') User restarted the server" >> "/var/log/httpd/actions.log"
                        logger -p user.info "'$username' restarted the server"
                        ;;
                esac
            esac
        echo "Location: menu.sh?username=$username"
        #echo "$(date '+%Y-%m-%d %H:%M:%S') User went back to main menu" >> "/var/log/httpd/actions.log"
        logger -p user.info "'$username' went back to main menu"
    done
fi
```

Manage_logs.sh

Manage_logs.sh is a simple script that displays (using a scroll-pane) all the lines in the `/var/log/httpd/actions.log` file where the word `apache` appears. Actions.log, as said before, is the file where each action of each user perform is being logged. They are being logged using the `syslog` messaging tool.

In each script when an action is performed, apart from executing that action it also sends a message with the logger command, with priority informational and facility user.

```
CONTENTS=$(grep "apache" "/var/log/httpd/actions.log")
```

Moreover in the `/etc/syslog.conf`, the following line is added:

```
user.info -/var/log/httpd/actions.log
```

This is used to store all the messages with facility user and priority informational into the `actions.log` file.

User_management.sh

User_management.sh displays at one side of the page the list of existing real users (no daemon users are displayed). On the other side it displays the two fields, one for the username and the other for the password and two respective buttons (one to add the user and the other to remove the user).

When the add button is pressed, it checks if both the password and username field contain data. It adds the user to the system and checks if it has been correctly added or if it already exists.

If the remove button is pressed, same way as the login, it will check if the user exists in the `/etc/passwd` file. Furthermore, we will get the hashed password and salt from that user. Additionally we will generate the hashed password of the entered one with the salt and compare them. If they are equal the user is removed. Consequently, if they are not equal, the user is not removed.

```
case $option in
  1)
    echo "Location: menu.sh?username=$username"
    logger -p user.info "$username went back to main menu"
    ;;
  2)
    if [ -n "$user" ] && [ -n "$pass" ]; then
      sudo useradd "$user" -m

      if [ $? == 0 ]; then
        echo "$user:$pass" | sudo chpasswd
        logger -p user.info "'$username' successfully added $user"
      else
        logger -p user.info "'$username' tried to add $user, but failed"
      fi
    fi
    ;;
  3)
    if grep -q "^$user:" /etc/passwd; then
      hashed_password=$(sudo grep "^$user:" /etc/shadow | cut -d':' -f2)

      salt=$(sudo grep "^$user:" /etc/shadow | cut -d':' -f3)
      hashed_pass=$(sudo openssl passwd -1 -salt "$salt" "$pass")

      if [ "$hashed_password" == "$hashed_pass" ]; then
        sudo userdel -r "$user"
        echo "Location: user_management.sh?username=$username"
        logger -p user.info "'$username' successfully deleted $user"
      else
        echo "Location: user_management.sh?username=$username"
        logger -p user.info "'$username' tried to remove $user, but password didn't match"
      fi
    else
      echo "Location: user_management.sh?username=$username"
      logger -p user.info "'$username' tried to remove a non existing user: $user"
    fi
    ;;
esac
```

Packet_filtering.sh

Packet_filtering.sh displays the 3 types of IPtables (Filter, NAT and Mangle) and it also displays a drop-down menu to select the type of table, different text fields to enter: the protocol, the source IP, the destination IP, the port, the action and a button to submit the entered data.

Once the button is pressed it gets the data, checks which fields has been entered, which have been left empty and adds the rule to the corresponding table with the fields entered.

```
rule=""
[ -n "$protocol" ] && rule="$rule -p $protocol"
[ -n "$source_ip" ] && rule="$rule -s $source_ip"
[ -n "$dest_ip" ] && rule="$rule -d $dest_ip"
[ -n "$port" ] && rule="$rule --dport $port"
[ -n "$action" ] && rule="$rule -j $action"
case $table in
    filter)
        sudo iptables -A INPUT $rule
        ;;
    nat)
        sudo iptables -t nat -A PREROUTING $rule
        ;;
    mangle)
        sudo iptables -t mangle -A INPUT $rule
        ;;
esac
logger -p user.info "$username added iptables rule to $table table: $rule"
;;
```

Lastly, to be able to display the tables and add rules the `iptables` command has been used. To use it, we followed the guide in BLFS book.

Manage_tasks.sh

Manage_tasks.sh displays a scroll-pane with the existing tasks for the user who logged in plus the different text fields to enter the minutes, hours, days, months, weekdays and the script to be executed followed by a button to create the task and another to delete it. When the button is pressed, the data is retrieved and the `fcrontab` command is used to either remove the task or add it.

As the `fcrontab` has been used to add, remove and display tasks, the `fcron` has been installed. To do so, the installation has been made following the steps from the BLFS book.

```
2) task=$(echo -e "$encoded_task" | sed 's/8\([0-9A-Fa-f]\{0-9A-Fa-f\}\)/\\x1/g')
cron_expression="$minute $hour $day $month $weekday"
(sudo -u $username fcrontab -l ; echo "$cron_expression $task") | sudo -u $username fcrontab -
logger -p user.info "$username" added the task: '$task' scheduled each $minutem minute, each $hour hour, each $day day and each $month month(weekday: $weekday)"
!!

3) task=$(echo -e "$encoded_task" | sed 's/8\([0-9A-Fa-f]\{0-9A-Fa-f\}\)/\\x1/g')
cron_expression="$minute $hour $day $month $weekday"
logger -p user.info "$username" removed the task $task scheduled each $minutem minute, each $hour hour, each $day day and each $month month(weekday: $weekday)"
sudo -u $username fcrontab -l | grep -v -F "$cron_expression $task" | sudo -u $username fcrontab -
logger -p user.info "$username" removed the task $task scheduled each $minutem minute, each $hour hour, each $day day and each $month month(weekday: $weekday)"
!!
```

Music_search.sh

Music_search.sh displays the contents of the file generated with the songs of the USB when a USB is connected to the system.

To be able to generate the file, the USB must be mounted first. To mount it, first a file has been created in `/etc/udev/rules.d/` and the line `SUBSYSTEM=="block", ACTION=="add", ATTRS{idVendor}=="0930", ATTRS{idProduct}=="6544", RUN+="/usr/local/bin/mount_usb %k"` has been added.

This line makes the system execute the file `/usr/local/bin/mount_usb %k` passing as argument the device when the system detects a USB with PID 6544 and VID 0930, which is the PID and VID of my USB.

Moreover the file `/usr/local/bin/mount_usb` has been created, given execution permission and inside the following lines have been added:

```
#!/bin/bash
sleep 2
mount /dev/$1 /mnt/usb-drive

if [ $? -eq 0 ]; then
    find /mnt/usb-drive -type f -name "*.mp3" > /mnt/songs.txt
    echo "MP3 files list created: /mnt/songs.txt"
else
    echo "Failed to mount the USB drive."
fi
```

A `sleep` has been used in order for it to not try to mount it before the USB has properly been connected. Hence, it is then mounted in the folder `/mnt/usb-drive`.

Finally, if the mount has been successfully, then it gets all files of type `.mp3` and stores its path in the `/mnt/songs.txt` file.

4. Observed Problems and Conclusion

During the implementation of the login a problem was found when changing the file type from an html to a sh. The file was called index, and when being an html, it was by default the default page when the URL was the localhost. But, after changing it to a sh file, and searching for the localhost it did not find anything. After some research, the DocumentRoot line, previously explained, was added to the httpd configuration.

In summary, the LFS project's web-based management system offers a secure and streamlined solution for remote server control. With a focus on simplicity and diverse functionalities, users can efficiently manage processes, monitor resources, and perform essential tasks. The project prioritizes transparency through log management and user authentication, providing a comprehensive and user-friendly approach to Linux system administration.