Labrador Intrusion Detection System

Documentation for version 0.8.2

Table of Contents

PREFACE

Labrador (also known as *Labrador Intrusion Detection* or *labrador-ids*) is a multiplatform tool developed to identify any unwanted modifications made in your system. It can be used as an integrity checker and as a host intrusion detection system (IDS). Labrador's differential is in it's unique set of available tests and response mechanisms, it's ease of use and it's flexibility, and it aims to be a *complete*, *free*, *multiplatform*, and *open-source* solution for detecting modifications and tamperings in files.

Labrador was designed from scratch to fill in a gap in current integrity checkers. While there are several great tools out there, most of them are either too intrusive/complicated for non-experienced users to try or too incomplete do fulfill people's security needs, whether they're advanced systems administrators or simply regular users concerned with their data.

Labrador was created with power and simplicity in mind. It focuses on several checks that you can make in a file or directory (folder) tree, such as checksum hashes, not allowing new or executable files in a given directory, among many others. Also, it's messages can be displayed in english, brazillian portuguese and spanish.

All rules are defined in a simple text file (the *rules file*) where you put them as cascating <tags> for activation/deactivation. Then all you need to do is tell the names of files and directories to be checked with the previously specified rules. All rules names were selected to be as clear as possible, so you don't have to spend hours memorizing them - A small glimpse at a rule tag will be enough for you to know exactly what it does (instead of some obsure "m", "J", "q" directives found in similar programs).

The program is released under the GNU General Public License, version 2.

The documentation below is provided in every Labrador release, in the file README.TXT. There is also a brazilian portuguese version available in LEIAME.TXT. The documentation in PDF (this file) and HTML are available only on the website:

http://labrador-ids.sourceforge.net

Please report any bugs/bugfixes, and any suggestions you may have on making Labrador a better software.

Thank you for using Labrador!

-b.

0 – QUICK REFERENCE

Installing:

Simply uncompress the .tar.gz (or .zip) file inside any given directory and you're set!

Creating a new database:

Edit the "rules.conf" file to choose which files (and which of the files' attributes) Labrador should monitor in your system. Afterwards, inside the directory created after the program's decompression, type:

```
./labrador.pl -n
```

Checking your system:

Just type, inside the directory created after the program's decompression:

```
./labrador.pl
```

Note: Depending on the obtained version, replace "labrador.pl" in the above commands for "labrador.exe" (Windows binary) or just "labrador" (Linux binary). In Windows systems, don't type the preceding "./" part.

1 - PRE-INSTALLATION

> **Note:** this step is only necessary if you're planning to use the original Perl file 'labrador.pl'. If you're planning to use one of the compiled versions (such as 'labrador.exe'), you may skip this step and go to section 2. Compiled versions of Labrador do ***NOT*** need Perl installed.

Labrador is a Perl program, so in order to run it you will need to have perl (version 5) installed in your system.

1.1 - BSD, GNU/Linux, Solaris, etc.:

Perl already ships inside most UNIX and UNIX-Like systems. To check if perl is correctly installed, open a terminal session and type "perl -v". If all went well, a message from the perl interpreter was shown on the screen. Otherwise, get the latest version of perl available at:

http://www.perl.org

1.2 - Microsoft Windows

To use Labrador in a Windows platform, you'll need to get a perl interpreter that runs on it. Try, for example, the one from ActiveState, which is free and available at:

http://www.activestate.com/Products/ActivePerl/

1.3 - Make sure required perl modules are installed

You should also install all Perl modules used by Labrador, which are:

- Digest::MD5
- Digest::SHA
- Digest::Haval256
- Digest::Whirlpool
- Digest::CRC
- Crypt::RIPEMD160
- Archive::Zip
- Crypt::CBC
- Term::ReadKey
- Getopt::Long
- Net::SMTP
- Win32::File (Windows users only)
- Win32::Console::ANSI (Windows users only)
- Term::ANSIColor (other systems)

Those can be easily installed manually via CPAN or PPM. To install Crypt::CBC, for example, through CPAN, simply type (as root)

```
# perl -MCPAN -e 'install Crypt::CBC'
```

As of version 0.8, Labrador also offers an easier option to install all the required modules. To do so, simply run the "perlcheck.pl" script located in the "/tools" directory inside the directory where Labrador was uncompressed. Whenever a module isn't found, the script will report it and attempt to automatically install the required module, as shown in the example below:

> **Note:** Don't forget to run the script as *"root"*, or the modules won't be installed correctly!

```
./perlcheck.pl
checking system for "File::Find"...FOUND!
checking system for "FindBin"...FOUND!
checking system for "Getopt::Long"...FOUND!
checking system for "File::stat"...FOUND!
checking system for "Digest::MD5"...FOUND!
checking system for "Digest::SHA"...FAILED!
  Would you like to install it (y/n)? y
```

If there's any module missing, don't forget to run the script once again after the needed module is installed to ensure that everything is installed correctly. In this case, the "perlcheck.pl" script will return the following message:

```
All set, and good to go! Enjoy Labrador! ;)
```

2 - INSTALLATION

Just uncompress Labrador's files to a local directory and you're all set!

3 - USAGE

Labrador must run for the first time in a recently installed system, or one that you're sure (***REALLY SURE***) that was not broken into.

3.1 - Creating a new database

The first step is to create a list of files and directories that you wish to monitor, and tell Labrador which "atributes" you want to check. This is done from Labrador's rule file, called "rules.conf" by default. If you want to give it a different name, make sure to pass the new name to Labrador when you run it. See section 5 of this document for more information on the rules file.

Once you have your own rules file, simply run:

```
labrador.pl -n
```

(you should also use the "-v" parameter to make Labrador give a more verbose output like shown below)

Note that in most cases you'll need administrator's privileges, or Labrador won't be able to analyze privileged files. Therefore, make sure you're running Labrador as root (in UNIX-like systems) or as Administrator (in Windows NT/2K/XP systems).

Also note that all the examples in this document use the original Perl file 'labrador.pl'. If you're using any other version of Labrador, simply replace 'labrador.pl' with the file name (like, for example, *'labrador.exe'* or *'labrador'*).

Labrador will then interpret the "rules.conf" file (or any other specified file) to generate a database file of your system in a file called "labrador.db". When it stops running, store the "labrador.db" file in a safe and write-protected media, such as a CD or floppy disk, and keep it safe. Do not forget to erase the local "labrador.db" file.

If you wish to encrypt the database, change the "labrador.conf" file accordingly. See section 4 of

this document for further information.

It is even possible to compress the database through the *"-c"* or *"--compress_db"* parameters.

During the creation of Labrador's database file, errors will be recorded in "labrador.err" file, and are likely to have happened due to the non existence of specified files or directories in your "rules.conf" file (which is a common thing if you used a generic rules file) or because Labrador didn't have permission to read certain files (and in this case you're probably running it from an unprivileged account). Either way, Labrador will simply skip the files that generated the errors and build a "labrador.db" file with the ones it could read.

If you simply want to list the files that the rules file will read without actually creating the database, maybe to see if a particular file will (or will not) be included in the list, just run:

```
labrador.pl -l
```
or
```
labrador.pl --list
```

which both will display all files that will be included in the database file. Note that arguments -n and -l cannot be used at the same time, causing Labrador to display an error message and quit.

3.2 - Checking your system

After creating a database file you can, whenever you want, run Labrador to check if your system has been modified (usually due to the action of malicious users or attacks to your system).

In order to do that, just run:

```
labrador.pl -f FILENAME
```

Where FILENAME is the full path of the "labrador.db" file created earlier. This can be something like:

```
labrador.pl -f /mnt/cdrom/labrador.db
```

in UNIX-like systems, or:

```
labrador.pl -f d:\labrador.db
```

in Microsoft systems.

If the "-f" parameter is omitted, the "labrador.db" file will be looked for in Labrador's directory.

Note that the "-v" parameter can be given in any of the steps before, and will make Labrador tell you everything that is being done.

3.3 - Logging the output

Every time Labrador detects a problem, a message shows up on screen indicating the file and which test acused the modification. Although this procedure is great for having an idea of how many

problems were identified, it may be dificult to follow all the messages shown on the screen. If you want to create a log for a performed verification, use the "--logfile FILE", as shown in the example below:

```
labrador.pl --logfile labrador.log
```

The command above will create the file "labrador.log" and append to the end of it all identified problems during verification. This means that the log file will continue to grow until you choose another name for it, or erase previous entries. Every addition to the log file is preceded by a header indicating the date and time of the check. For instance:

```
##############################################
# Labrador's Log File
# (entry created in Thu Feb 24 01:54:09 2005)
##############################################

File "/var/www/htdocs/index.html" - File could not be opened for reading (deletion or
wrong permissions)
```

3.4 - The "--rootdir" parameter

Another useful parameter is the '--rootdir' (or simply '-r') option, which sets an alternative root directory for eather creating or checking a database file. For example, if an attacker loaded some sort of kernel module that makes your system "lie" about its files atributes, you can run Labrador from a "safe" system, with the suspicous system mounted in /mnt/hdb (in a UNIX-like system) or in "D:\" (in a Windows system), for example, and then run:

```
labrador.pl -r /mnt/hdb
```
(or "**-r D:**", for the Windows example)

This will make Labrador understand that "/bin" in the database file is now "/mnt/hdb/bin". In the Windows example, Labrador will know that "\WINNT\SYSTEM32" is now "D:\WINNT\SYSTEM32". This is also really useful for Windows systems that change drive letters a lot. This way you can create your "rules.conf" rules without the drive letter, and add it later with the "-r" parameter.

4 - THE "LABRADOR.CONF" FILE

The 'labrador.conf' file can be changed to contain all your preferences. It can control the following properties:

4.1 - General Settings:

4.1.1 - Language:

Define the language of Labrador's messages by changing the variable "lang". Available languages:

```
lang=pt_br     # messages in brazilian portuguese
lang=us        # messages in english
lang=es        # messages in spanish
```

4.1.2 - Verbose mode

You can toggle verbose output (e.g. messages on screen saying what is currently being added to the database or checked) by putting the following string on your labrador.conf file:

```
verbose=yes
```

4.1.3 - Log file

By changing the "logfile" entry, it is possible to set which log file name
to write to. Default is "labrador.log":

```
logfile=/path/to/my/labrador.log
```

4.1.4 - Database

Labrador creates/searches its database in the 'labrador.db' file inside the directory where the program is. This location can be redefined with:

```
db_file=/path/to/my/labrador.db
```

4.1.5 - Rules file

It is possible to define the rules file that Labrador should use, by modifying the "rules_file" entry:

```
rules_file=/path/to/my/rules.conf
```

4.1.6 - Error file

It is possible to define the file in which Labrador should record any errors during the creation of the database:

```
error_file=/path/to/my/labrador.err
```

4.1.7 - Alternative Root Directory

You can also use the "rootdir" entry to set an alternative root directory for running the check (if drive is mounted somewhere else). The line below:

```
rootdir=/mnt/suspicious_hd
```

will append the given directory ("/mnt/suspicious_hd") to every file verified. This means that, instead of checking for, e.g., "/bin", Labrador will check for "/mnt/suspicious_hd/bin".

4.2 - Email:

4.2.1 - Mail Alert

Labrador can send its reports through email. To activate email sending, give the "yes" value to the "mail_alert" variable:

```
mail_alert=yes
```

It is even possible to configure Labrador to send email messages only when it finds a problem. To do that, set "mail_alert" as:

```
mail_alert=onerror
```

Note, however, that the following configurations must also be set in order to the correct email handling:

4.2.2 - SMTP Server

This defines the location of the smtp server to be used (default: localhost)

```
smtp_server=localhost
```

4.2.3 - Destination Email

Use the "rcpt_to" entry to choose the destination email (default: postmaster@localhost)

```
rcpt_to=postmaster@localhost
```

4.2.4 - Email of Origin

The "mail_from" entry sets the email of origin (default: labrador@hostname)

```
mail_from=labrador@hostname
```

4.3 - Database cryptography and compression

4.3.1 - Encrypt

It is possible to encrypt the database by putting the "encrypt" variable with the "yes" value.

```
encrypt=yes
```

But keep in mind that you'll have to choose a cipher (see below)

4.3.2 - Cipher

The "cipher" entry lets you choose a symmetric cryptography cipher. Note that in order to do so you will have to install the respective cipher module via CPAN or PPM. Virtually any symmetric cipher compatible with the "Crypt::CBC" module can be used. Some of them (and their respective cpan modules) are:

- DES (Crypt::DES)
- IDEA (Crypt::IDEA)
- Blowfish (Crypt::Blowfish)
- CAST5 (Crypt::CAST5)
- Rijndael (Crypt::Rijndael)
- Twofish (Crypt::Twofish2)
- Anubis (Crypt::Anubis)

Once the module is installed, simply add its name to the "cipher" field. Example:

```
cipher=Blowfish
```

Note: The pre-compiled binaries "labrador" (NIX) and "labrador.exe" (Windows) already have the Blowfish and DES ciphers. To use any other methods you need to use the Perl version.

4.3.3 - Compression

When set to "yes", The "compress_db" entry will create a zip file from the database (default: no).

```
compress_db=no
```

4.4 - Backup Options

4.4.1 - Backup Filename

It is possible to use the "backup_file" field to automatically select a different path or name for the backup file to be created or read. (default is "backup.lab")

```
backup_file=/mnt/cdrom/labrador_backup.lab
```

4.4.2 - Quarantine

The "quarantine" entry, when set to "yes", will always create a backup copy of every file overwritten by a restored backup. (default: no)

```
quarantine=yes
```

4.4.3 - Quarantine Filename

Change this option to use a filename or path of your choice for the quarantined file. (default: quarantine.zip inside labrador's directory)

```
quarantine_filename=/root/labrador_quarantined.zip
```

4.4.4 - Auto-restore

The "auto_restore" entry can be used to automatically restore all tampered files that have a backup. Note that this option should be used with *EXTREME CAUTION* as it will overwrite every modified file with the original backup without asking for confirmation. It is recommended that you use this option always together with the quarantine option above. (default: no)

```
auto_restore=yes
```

5 - THE "RULES.CONF" FILE

The rules file has an intuitive format based on XML tags, where all you need to do is tell which atributes you want to register and, on the following lines, add a list of files and directories (one per line) to which the selected rules should be used. When you don't want the set attribute to be registered anymore, just end it's tag (by repeating the tag with a "/" before the name, as shown below).

There are some example files for different platforms available in Labrador's official distribution:

File	Description
rules-OpenBSD.conf	example rules for OpenBSD (and other unices) environments
rules-Linux.conf	example rules for Linux environments
rules-winNT2K.conf	example rules for Windows NT/2K environments
rules-winXP.conf	example rules for Windows XP environments

To use them, just rename the desired file to "rules.conf", or pass it as a parameter through the "rules_file" item from Labrador's configuration file.

> **WARNING:** The example files are there just as an EXAMPLE! You should ***not*** rely on them to secure your systems.

5.1 - Accepted tags

The Labrador rules file accepts the following tags:

Activation	Deactivation	Description
<md5>	</md5>	creates MD5 (128 bits) checksum of files
<sha1>	</sha1>	creates SHA-1 (160 bits) checksum of files
<sha224>	</sha224>	creates SHA-224 (224 bits) checksum of files
<sha256>	</sha256>	creates SHA-256 (256 bits) checksum of files
<sha384>	</sha384>	creates SHA-384 (384 bits) checksum of files
<sha512>	</sha512>	creates SHA-512 (512 bits) checksum of files
<haval>	</haval>	creates haval256 (256 bits) checksum of files
<whirlpool>	</whirlpool>	creates whirlpool (512 bits) checksum of files
<ripemd160>	</ripemd160>	creates RIPEMD-160 (160 bits) checksum of files
<crc32>	</crc32>	creates CRC-32 (32 bits) checksum of files

<crc16>	**</crc16>**	creates CRC-16 (16 bits) checksum of files
<crc8>	**</crc8>**	creates CRC-8 (8 bits) checksum of files
<crcccitt>	**</crcccitt>**	creates CRC-CCITT (16 bits) checksum of files
<mode>	**</mode>**	registers file type and permissions
<uid>	**</uid>**	registers user id of file's owner
<gid>	**</gid>**	registers group id of file's owner
<nlink>	**</nlink>**	registers number of (hard) links of file
<inode>	**</inode>**	registers the file's inode number
<mtime>	**</mtime>**	registers the file's last modification time
<atime>	**</atime>**	registers the file's last access time
<ctime>	**</ctime>**	registers the file inode's last modification time
<size>	**</size>**	registers the total file size, in bytes
<nblocks>	**</nblocks>**	registers the number of blocks allocated to file
<dev>	**</dev>**	registers the device number of filesystem for file
<grow>	**</grow>**	registers that the file size can only increase
<nonew>	**</nonew>**	registers that the directory must not contain new files (created after the database was made)
<nodel>	**</nodel>**	registers that no files can be deleted from the directory
<nosuid>	**</nosuid>**	registers that the directory must not contain suid files
<nosgid>	**</nosgid>**	registers that the directory must not contain sgid files
<noexec>	**</noexec>**	registers that the directory must not contain executable files
<nobinary>	**</nobinary>**	registers that the directory must not contain binary files
<notext>	**</notext>**	registers that the directory must not contain text files
<nohidden>	**</nohidden>**	registers that the directory must not contain hidden files
<nosymlink>	**</nosymlink>**	registers that the directory must not contain symbolic links
<reset>	**-**	unset ALL tags

Note that the <reset> tag is special and equivalent to unsetting all others. This means that writing <reset> is the same as writing </md5> </suid> </...> for all other tags. This allows you to easily regain control of your file in case you lost yourself with the cascading rules, or if you just don't want to close each and every tag.

Also note that some of the tags are not available or behave differently in specific systems. Please check section 6 of this document for a list of known issues concerning non *NIX operating systems.

Important note: Try to resist the temptation of using all tests at the same time for a given file or directory. Some checks do not get along very well together, for instance:

<grow> x <md5> (or any other hash)
if we expect a file to grow, we shouldn't expect it to keep it's same signature in each check;

<ctime> x <atime> (in *NIX environments)
in order to perform some checks, Labrador reads the requested files from the system. If the access time (<atime>) check is in effect, Labrador restores the original value after it accesses the file in question (otherwise the access time would change and Labrador would warn about it in future runs). The problem is that, after restoring the file's last access time, the file's inode's last modification time (<ctime>) changes, and it can't be restored in the "userspace".

5.2 - Inserting comments

You can also put comments in your rules file, which should be lines beggining with a "#" character. It's even possivel to put comments in lines with rules or files, as Labrador ignores everything after the "#" character.

Although tags can be put together on the same line, it is not possible (yet) to put more than one file or directory per line, or to put tags, files and directories in the same line.

5.3 - Wildcards, Recursion and Exclusion

The rules file also accepts special wildcards: if you place a "*" after a directory, like

```
/usr/local/ *
```

Labrador will scan all files within that directory. This will also happen if you simply put a directory to be scanned, like

```
/usr/local
```

If you put a "-r" (not to be confused with the "-r" parameter passed in the command line) after a directory, like

```
/etc/ -r
```

Labrador will scan the entire directory plus every file in every subdirectory of the one specified. You can use the "*" and "-r" rules above together with the exclude rule, which disables scanning of a specified file or directory followed by an exclamation "!". The rules:

```
/usr/local/ !
/usr -r
```

would scan the entire /usr directory plus its subdirectories, EXCEPT the ones inside "/usr/local/".

5.4 - Examples

Below are some examples of a rules.conf file:

```
<md5> <uid>

  /bin/bash
  /bin/su

</md5>

  /etc/shadow

</uid>
```

The above lines will make Labrador register the md5 checksum and the user id of files "/bin/bash" and "/bin/su", but only the user id of file "/etc/shadow". The order in which you place the tags is not relevant, so if the tags were "<uid> <md5>" instead of "<md5> <uid>", the result would be the same.

Note that files and directories listed in the rules file can follow the standard of the system where Labrador is running. In Microsoft systems, for example, we could find the following code inside "rules.conf":

```
<mode>

   \winnt\system32\config\system.dat
   \winnt\system32\config\user.dat

<mtime> <md5>

   \winnt\system32\config\SAM

</md5> </mode>

   \winnt\system32\cmd.exe

</mtime>
```

The code above will tell Labrador to register type and permissions of files "system.dat" and

"user.dat". Next, besides type and permissions, Labrador should also record the time of the last modification and the MD5 hash of file "SAM". Finally, it registers just the last modification time of file "cmd.exe".

Although you could put the drive letter in the rules file, for the sake of portability it is best if you leave it out. In this case, you need to tell Labrador in which drive the system is with the "-r" parameter, for example:

```
labrador.pl -r c:
```

This way you can use the same "rules.conf" rules file in other Windows systems even if they are in a different drive letter. The only reason *NOT* to do this (imho) is if the files you wish to record are in two or more partitions. For example, in the same system you might want to register both "C:\WINNT" and "D:\MYSTUFF". Even so, you can create a rules file for your entire "C" drive and another one for "D" drive.

If you want to run Labrador against a configuration file with a name other than "rules.conf", you should use the "-f" parameter, telling the name of your file. For example:

```
labrador.pl -n -f myrulesfile.conf
```

5.5 - The Special <execute> Tag

As of Labrador version 0.8.0, there's a special tag called <execute>, which can run a program or script of your choice whenever a rule is broke. For example:

```
<md5>

<execute="/root/help.sh">
  /sbin

<execute="/root/trouble.pl --labrador">
  /var/www/htdocs
```

The code above will test MD5 hashes for both "/sbin" and "/var/www/htdocs" directories. But if Labrador detects a problem while checking hashes for the "/sbin" files, it will immediately run the "/root/help.sh" file, while still running the test. And if, while continuing the test, Labrador find a tampered file inside "/var/www/htdocs", it will run the "/root/trouble.pl" file passing the "--labrador" parameter as written in the example above - even if "/root/help.sh" hasn't terminated yet!

This tag was created to help respond to problems immediately after they are detected.

5.6 - Creating your own tag

With Labrador, it's possible to create your own tag in a quick and easy way, allowing the creation of a cleaner and easier to understand rules file. To do so, just set one or more existent tags to a new tag

of your choice, for example:

```
<my_tag> = <md5> <uid> <gid>

<other_tag> = <noexec> <my_tag>
```

After the new tag is declared, whenever it's used in your rules file,
it will be interpreted as the group of original tags. As such, writing:

```
<my_tag>
   /bin
```

is the same as writing:

```
<md5> <uid> gid>
  /bin
```

6 - BACKING UP AND RESTORING FILES

6.1 - The Special <backup> Tag

Another useful feature of Labrador is the possibility of creating backup copies of files in your system. This is done through the <backup> tag inside the rules file. All files between the <backup> and </backup> tags will be compressed and archived in 'backup.lab' file. For instance, the rule:

```
<backup>
   /var/www/htdocs/*
</backup>
```

will register the files inside the '/var/www/htdocs/' directory, with all the options defined before this in the rules file (such as <md5>, <mode>, etc) and will create a backup copy of all these files inside 'backup.lab'.

It is even possible to choose a different path and/or file name for the backup file through the '-b' or '--backup_filename' parameter, as shown in the example below:

```
   labrador.pl -n -b FILE
```
or
```
   labrador.pl -n --backup_filename FILE
```

where 'FILE' is the new backup file name. Note that we have also used the '-n' parameter to indicate that we are creating a new database from the current rules file.

Once the database (and backup file) is created, if Labrador detects in future verification runs that a test for a given file has failed, it will not only report the problem to the user but also check if a backup copy of the file was made and, if so, ask the user if he/she wants to restore the file. If so, the restauration is done automatically.

6.2 - Restoring Automatically

If you wish to automatically restore all the modified files without Labrador asking, simply pass the '-e' or '--restore' parameter like this:

```
   labrador.pl -e
```
or
```
   labrador.pl --restore
```

Note, however, that if the backup file is in a different path or with a name other than 'backup.lab', you must tell the new path/name through the '-b' parameter seen earlier.

This Labrador feature is ideal for protecting static web pages from defacements, for example. Simply include Labrador in the crontab (in *NIX systems) or in some kind of task scheduler (in other systems), to perform periodical checkings using a Labrador rules file specific for the web page.

This way, if the page get defaced, it will be restored as soon as the next run of Labrador is performed.

> **ATENTION:** Although Labrador restores files to their original state, the simple fact that they were modified means that someone may have invaded your system. And this means that they may do it again. This option must be used only to keep the integrity of the file, and must not be mistaken with a false sense of security. It is recomended, therefore, that the administrator use Labrador always with log options and check them frequently if he/she chooses to use Labrador periodically through an automatic tool. Also note that the modified file will be lost, and that it *MAY* have been intentionally modified by you or by someone authorized, that simply did not know that the file was being monitored by Labrador. Computer forensics procedures may also be invalid or hardened if the modified files are erased. So, use the '--restore' parameter with extreme caution.

6.3 - Quarantine

While automatic restore of backed up files is a nice feature, sometimes you wish to keep the modified file for analysis. Labrador lets you do it also automatically via the "quarantine" parameter, that can be set by command line or via the "labrador.conf" file (see section.4.4.3). The following command:

```
./labrador.pl -e -q
```

will restore all modified files with backup (-e) but also will keep the modified version inside the file "quarantine.zip", which is a regular "zip" file that can be accessed by any compression program. You can also change the name and path of the quarantine file via the "quarantine_filename" parameter:

```
./labrador.pl -q --quarantine_filename=/root/labrador-modified.zip
```

this will store the quarantined items in the "/root/labrador-modified.zip" file. Note that you don't have do pass the "-e" parameter whenever you're using quarantine. The command above, without the "-e", will prompt the user to restore the backup or not, and all replaced items will be kept in the specified quarantine file.

7 - KNOWN PORTABILITY ISSUES

Labrador has been successfully tested on the following platforms:

- GNU/Linux
- Microsoft Windows (2000, XP, 9x, Vista)
- OpenBSD
- MacOS X

Labrador was built from the start with portability in mind. Even so, when it comes to analysing a filesystem, each OS has its own way of doing things. So you should be aware of what works and what doesn't in your own system when you use Labrador.

Below there's a list of known issues concerning file attributes in different operating systems. Please note, however, that I did not have the opportunity to test Labrador in all these platforms, and so most of the issues below were gathered only from feedbacks of the perl community concerning some of the functions that Labrador uses. Untested platforms were marked below with a '*', and I'd really appreciate any feedback on whether the statements below are true or not.

7.1 - Microsoft Windows

- Windows users will not benefit from 'inode', 'nblocks' and 'device' tags, as they are meaninless in FAT and NTFS filesystems.

- Also, the 'ctime' tag will record the file creation time, instead of the inode modification time as in *NIX systems.

*7.2 - Mac OS**

- The 'mtime' and 'atime' tags register the exact same thing.

- Also, the 'ctime' tag will record the file creation time, instead of the inode modification time as in *NIX systems. But it is not supported on UFS (A filesystem of Mac OS X).

*7.3 - VMS**

- The 'device' and 'inode' tags are not necessarily reliable, and as such you should only use them if you know what you're doing.

*7.4 - RISC OS**

- The 'mtime', 'atime' and 'ctime' tags all return the file's last modification time ('mtime').

- Also, the 'device' and 'inode' tags are not necessarily reliable,
  and as such you should only use them if you know what you're doing.

*7.5 - OS/2 (now eComStation)**

- OS/2 and eCS users will not benefit from 'inode', 'nblocks' and
  'device' tags.

8 - THE "LABRADOR.DB" FILE

This is Labrador's database file, created every time the '-n' parameter is passed to the program. It's simply a text file containing the properties recorded from every file in the rules file, with every field separated by a pipe '|'.

It's first line contains a special string, 'LABRADOR-DB-FILE', as a way to prevent you from loading a non-database file by mistake (believe me, it has happened a lot).