p, A password manager

Giovanni Cherubin

p, A password manager

p is a very simple command line password manager. Its goal is to be an easy to read and mantain password manager, with some care to users security. Passwords are all stored by *p* in a local file, which is encrypted by using GnuPG public key encryption. A *master password*, the GnuPG private key password, ensures that an attacker, managing to have access to the encrypted storage and to the private key, cannot immediately obtain all the passwords. A weak *master password* although will not ensure he won't be able to get them at some point. Every password is associated to a label, which should represent the service the password is used on (e.g.: 'twitter', 'freenode', ...). When asked for a password, *p* copies it to the user clipboard in order to prevent shoulder surfing. *p* is easily scriptable. For example, a user may want to periodically check that there are no passwords older than 3 months. This can be achieved by using Unix cron, and performing a test on the output of:

```
$ p -l
```

which returns a list of the labels, together with their age in days.

p does not guarantee that an attacker controlling your computer will not be able to get all your passwords: a simple keylogger used while you type the master passphrase would do the job for him. However, a keylogger would be probably able to steal your passwords anyways while you type them...

p code was designed to be as simple as possible, so to enable peer reviews, a so that potentially anyone who understands a bit *bash* can check it does what it says before using it.

NOTE: this is still a beta version, use it if you want to test it, but keep your passwords also stored somewhere else, just in case...

Requirements

p is a bash script. It can be potentially used on every system supporting bash. The system should also have a copy-to-clipboard utility, as explained later, and

GnuPG [1] installed. Also 'awk' and other common *nix utilities should be present on the system.

Installation

After satisfying the requirements, the script `p.sh` can be put in any directory, and linked to an executable path, by doing for example:

```
# ln -s /full/path/to/p.sh /usr/local/bin/p
```

and then called by

```
$ p
```

Before this, please create a directory for the configuration file `p.cfg` and put it there. We recommend:

```
$ mkdir ~/.p/
$ cp p.cfg ~/.p/
```

else you'll have to change something in the configuration. We suggest to create a new GPG identity called 'pmanager', by:

```
gpg --gen-key
```

protected by a password. This password will be asked every time you need to read data from the storage file.

Configuration

Default configuration `~/.p/p.cfg` should be now present. A few variables from it must be checked before execution:

CMD_COPY

Must be the command line program to copy to clipboard (e.g.: *pbcopy* in OSX, *xclip* under *X*, *gpm* for Linux in terminal mode). A more complete list of these programs was given in [2]. This variable is set, by default, to *pbcopy*.

GPG_ID

The identity to set for this variable must be a prefix of the new created identity. In the suggested case it should be 'pmanager'.

STORE_ENC

This is the local file in which the encrypted data will be put. The default is `~/.p/store.gpg`, and it can be left as it is.

STORE_PLAIN

This is the local temporary file in which the plain text data will be put. The default is `~/.p/store`, and it can be left as it is.

Quick run

Here's a quick overview of the *p* commands you may want to use:

```
# Show help
$ p
# Add a new password
$ p -a twitter
# Show a stored password (insert GPG passphrase when prompted)
$ p twitter
# Remove a password
$ p -r twitter
# Modify a password
$ p -m twitter
# List labels and their passwords age
$ p -l
```

Issues

Using OSX, tmux and pbcopy

If you're an OSX user using tmux, you will probably not be able to use *pbcopy*, and thus the copy-to-clipboard *p* functionality. Well, there's a solution: [3].

Temporary file

This program is using for now a temporary file, `~/.p/store` as default. Now, an attacker with user permissions may do something like: $ while [1]; do [[-e "~/.p/store.gpg"]] && cp test evil; done which would allow him to get all the stored password in plaintext when the user decrypts the file. I'm not sure this is a great risk (the assumption of an attacker having user permission is dangerous itself), but I believe this can be solved by only using pipes within *p* code (no temporary file). I'm not sure about this either:)

References

[1] https://www.gnupg.org/

[2] http://stackoverflow.com/a/750466/1230980

[3] http://superuser.com/a/413233