NAME

Tomb – the Crypto Undertaker

SYNOPSIS

tomb [options] command [arguments]

DESCRIPTION

Tomb is an application to manage the creation and access of encrypted storage files: it can be operated from commandline and it can integrate with a user's graphical desktop.

Tomb generates encrypted storage files to be opened and closed using their associated keys, which are also protected with a password chosen by the user. To create, open and close tombs a user will need super user rights to execute the tomb commandline utility.

A tomb is like a locked folder that can be safely transported and hidden in a filesystem; it encourages users to keep their keys separate from tombs, for instance keeping a tomb file on your computer harddisk and its key file on a USB stick.

COMMANDS

dig Generates a file that can be used as a tomb and will occupy as much space as its desired initial size, the unlocked *.tomb* file can then be locked using a *.tomb.key*. It takes a mandatory option which is the *--size* in megabytes (MiB). This generation is relatively simple: its a data dump (dd) of low-quality random data (from /dev/urandom) and does not require root privileges.

forge Creates a new *key* and prompts the user for a *password* to protect its usage. This operation requires high quality random data (from /dev/random) which can take quite some time to be gathered on a server: it works better on a desktop where the mouse can be moved around for entropy.

lock Initializes and locks an empty tomb (made with *dig*) using a key (made with *forge*), making it ready for usage. After this operation, the tomb can only be open in possession of the key and knowing its password. As in any other command requiring a key, the option *-k* should be used to specify a key file. This operation requires root privileges to loopback mount, format the tomb (using LUKS and Ext4), then set the key in its first LUKS slot.

open Opens an existing *.tomb* (first argument) using a key (*-k*), if a second argument is given it will indicate the *mountpoint* where the tomb should be made accessible, else the tomb is mounted in a directory inside /media. The option *-o* can be used to pass mount(8) options (default: rw,noatime,nodev).

list List all the tombs found open, including information about the time they were opened and the hooks that they mounted. If the first argument is present, then shows only the tomb named that way or returns an error if its not found.

index Creates or updates the search indexes of all tombs currently open: enables use of the *search* command using simple word patterns on file names. Indexes are created using mlocate updatedb(8) and stored in a file inside the tomb's root. To avoid indexing a specific tomb simply touch a *.noindex* file in its root.

search Searches through all tombs currently open for filenames matching one or more text patterns given as arguments. Search returns a list of files found in all open tombs on which the *index* command

was run at least once.

close Closes a currently open tomb. If more tombs are open, the first argument should be used to specify the name of the tomb to be closed, or *all* to close all currently open tombs. This command fails if the tomb is in use by running processes (to force close, see *slam* below).

slam Closes a tomb like the command *close* does, but it doesn't fails even if the tomb is in use by other application processes: it looks for them and violently kills –9 each of them. This command may provoke unsaved data loss, but assists users to face surprise situations.

passwd

Changes the password protecting a *key* file specified using *-k*. The user will need to know the key's current password, then its content will be decoded and reencoded using the new one. This action can't be forced if the current password is not known. If the key file is broken (missing headers) this function also attempts its recovery.

resize Increase the size of a tomb file to the amount specified by the *--size* option in megabytes (MiB). Full access to the tomb using a key (*-k*) and its password is requires. Tombs can only grow and can never be made smaller. This command makes use of the cryptsetup resize feature and the resize2fs command: its much more practical than creating a new tomb and moving everything into it.

bury Hides a tomb key (*-k*) inside a *jpeg image* (first argument) using *steganography*: the image will change in a way that cannot be noticed by human eye and hardly detected by data analysis. This option is useful to backup tomb keys in unsuspected places; it depends from the availability of *steghide*.

exhume

This command recovers from jpeg images the keys that were previously hidden into them using *bury*. Exhume requires a key filename (*-k*) and a *jpeg image* file (first argument) known to be containing a key. If the right key password is given, the key will be exhumed. If the password is not known, it is very hard to verify if a key is buried in any image or not.

OPTIONS

-s <*MBytes*>

When digging or resizing a tomb, this option must be used to specify the *size* of the new file to be created. Units are megabytes (MiB).

-k <*keyfile*>

When opening a tomb, this option can specify the location of the key file to use. Keys are created with the same name of the tomb file adding a '.key' suffix, but can be later renamed and transported on other media. If <*keyfile*> is "-" (dash), it will read it from stdin.

--kdf <*seconds*>

Activate the KDF feature against dictionary attacks when creating a key: forces a delay of <*seconds*> every time this key is used. Floating point values are accepted, default is 1.

-n Skip processing of post-hooks and bind-hooks if found inside the tomb. See the *HOOKS* section in this manual for more information.

-o Manually specify mount options to be used when opening a tomb instead of the default *rw,noatime,nodev*. This option can be used to mount a tomb read-only (ro) to prevent any modification of

its data, or to experiment with other settings (if you really know what you are doing) see the mount(8) man page.

-f Force flag, currently used to override swap checks, might be overriding more wimpy behaviours in future, but make sure you know what you are doing if you force an operation...

-h Display a help text and quit

-v Display version and quit

-q Run more quietly

-D Print more information while running, for debugging purposes

--no-color
Don't use colors; useful for old terminals or integration in other scripts parsers

HOOKS

Hooks are special files that can be placed inside the tomb and trigger actions when it is opened and closed; there are two kinds of such files: *bind-hooks* and *post-hooks* can be placed in the base root of the tomb.

bind-hooks
This hook file consists of a simple two column list of files or directories inside the tomb to be made directly accessible inside the current user's home directory. Tomb will use the "mount –o bind" command to bind locations inside the tomb to locations found in $HOME so in the first column are indicated paths relative to the tomb and in the second column are indicated paths relative to $HOME contents, for example:

```
mail        mail
.gnupg       .gnupg
.fmrc       .fetchmailrc
.mozilla     .mozilla
```

post-hooks
This hook file gets executed as user by tomb right after opening it; it can consist of a shell script of a binary executable that performs batch operations every time a tomb is opened.

PRIVILEGE ESCALATION

The tomb commandline tool needs to acquire super user rights to execute most of its operations: to do so it uses sudo(8), while pinentry(1) is adopted to collect passwords from the user. Tomb executes as super user only when required.

To be made available on multi user systems, the superuser execution of the tomb script can be authorized for users without jeopardizing the whole system's security: just add such a line to */etc/sudoers*:

username ALL=NOPASSWD: /usr/local/bin/tomb

SWAP

On execution of certain commands Tomb will complain about swap memory on disk when that is presend and *abort if your system has swap activated*. You can disable this behaviour using the *--force*. Before doing that, however, you may be interested in knowing the risks of doing so:

- During such operations a lack of available memory could cause the swap to write your secret key on the disk.
- Even while using an opened tomb, another application could occupy too much memory so that the swap needs to be used, this way it is possible that some contents of files contained into the tomb

are physically written on your disk, not encrypted.

If you don't need swap, execute *swapoff -a*. If you really need it, you could make an encrypted swap partition. Tomb doesn't detect if your swap is encrypted, and will complain anyway.

EXAMPLES

- Create a 128MB large "secret" tomb and its keys, then open it:

  tomb dig -s 128 secret.tomb

  tomb forge secret.tomb.key

  tomb lock secret.tomb -k secret.tomb.key

  tomb open secret.tomb -k secret.tomb.key

- Open a Tomb using the key from a remote SSH shell, without saving any local copy of it:

  ssh user@my.shell.net 'cat .secrets/tomb.key' | tomb open secret.tomb -k -

- Create a bind hook that places your GnuPG folder inside the tomb, but makes it reachable from the standard $HOME/.gnupg location every time the tomb will be opened:

  tomb open GPG.tomb -k GPG.tomb.key

  echo ".gnupg .gnupg" > /media/GPG.tomb/bind-hooks

  mv ~/.gnupg /media/GPG.tomb/.gnupg && mkdir ~/.gnupg

  tomb close GPG && tomb open GPG.tomb -k GPG.tomb.key

- Create an exec post hook that launches a Firefox browser every time the tomb will be opened, keeping all its profile data inside it:

  tomb open FOX.tomb -k FOX.tomb.key

  touch /media/FOX.tomb/post-hooks

  chmod +x /media/FOX.tomb/post-hooks

  cat <<EOF >> /media/FOX.tomb/post-hooks

  #!/usr/bin/env bash

  if ["$1" == open]; then

  firefox -no-remote -profile $(dirname $0)/firefox_prof &

  fi

  EOF

BUGS

Please report bugs on the tracker at

Get in touch with developers via mail by subscribing the "crypto" mailinglist on *http://lists.dyne.org* or via the #dyne chat channel on *https://irc.dyne.org*.

AUTHORS

Tomb is designed, written and maintained by Denis Roio aka Jaromil.

Tomb includes code by Anathema, Boyska and Hellekin O. Wolf.

Tomb's artwork is contributed by Jordi aka Mon Mort

Testing and reviews are contributed by Dreamer, Shining, Mancausoft, Asbesto Molesto and Nignux.

Cryptsetup was developed by Christophe Saout and Clemens Fruhwirth

COPYING

AVAILABILITY

The most recent version of Tomb sourcecode and up to date documentation is available for download from its website on *http://tomb.dyne.org*.

SEE ALSO

cryptsetup(8)

GnuPG website on http://www.gnupg.org

DM-Crypt website on http://www.saout.de/misc/dm-crypt

LUKS website, http://code.google.com/p/cryptsetup