

Mobi 2.0

Mobi stands for “My Own Backup Implementation”, so it basically is my own idea of how an ideal linux backup should work.

You can read all about my first aim on creating the “perfect” (to me) backup utility :) However, requirements have changed over time and it is now time to find a new backup solution.

Mobi 2.0 will probably be a complete new set of scripts and it won't contain much or anything of the first Mobi script I wrote.

Why a complete new MOBI?

The main reason for a complete re-write of my old Mobi tool is that I want to better protect my data from ransomware and similar attacks as well as targeted hacking attacks, where a hacker (or a group of hackers) manually hijack a server and then try to cause damage to the owner of the server by messing with the data on it.

Mobi is not very secure in this regard, as it usually runs on a backup server that needs full root access to all backup clients it should pull backups from. So if someone hijacks the Backup server that person will automatically have password-less root access to any of the Linux systems it backs up, which is really bad to be honest!

On my private server, mobi is running on the client server itself and the backup is stored to another set of local disks.. that's also not very good, as ransomware would then encrypt both my data and my backup at once, so even the backup will be rendered useless.. even worse, since unchanged files between backups are hardlinked instead of copied, encrypting all the backups will be extremely fast, as one only has to encrypt each version of a file once.

Mobi does a great job to protect against accidental data loss, data loss due to hardware issues like multiple disk failures, loss of complete raidsets etc. and also in cases where the client server is hacked but the backup server is not.. since the backup is completely controlled from the backup server, there isn't anything a hacker can do on a client server (the one being backed up) to mess up the backup from there.

Since there have recently been an increasing number of reports of targeted hacking attacks on companies in my vicinity (meaning, switzerland in general, or simliar fields of operation, customers or direct compteditors of customers etc.) I realized it's time to re-think how i do backups and to set new goals as far as security goes.

the main goal of Mobi was to be as portable and simple as possible and to provide incremental backups where i had full snapshots of each backup in a simple folder structure to facilitate restores and make browsing of backups easy.

Design Goals for Mobi 2.0

- No more root access should be necessary between any of the involved machines!

- untrusted client machines -> we have to assume, that the machine we are creating a backup for could be hacked, so we have to make sure, that not even root on a client machine could delete or tamper with previous benchmarks. We will assume however, that the retention time for the backups is longer than the time the client machine has been hacked, meaning, we don't need to detect if the client is hacked or not.. we will continue to make backups for as long as the hacker wants to and hope that the admin of the hacked client will notice the attack before his oldest backup is purged from the repository
- untrusted servers: the backup server can't be trusted.. we have to assume that the backup server could be hacked and have to take precautions, that this does not lead to the client servers being hacked as well as a result.. for that reason, the backup server should not be allowed to login or execute anything on the client machine. also the backup target should not hold sensible information in an unencrypted format (such as /etc/passwd files, ssh keys etc. which may facilitate hacking the client servers)
- backups of a backup should be possible and should follow the same principles as mentioned above: Mobi 2.0 should at least make it possible, to create a secondary backup from the backup server to an offsite backup server, so that a local backup can be kept in a local network and a remote backup can be kept for disaster recovery like after a fire or water damage or similar. the secondary backup may lag slightly behind the main backup. It should be created by copying data from the backup server to the secondary backup server, not by just creating a second backup from the client directly.
- shortest expected backup intervals are daily backups.. however, it wouldn't hurt if shorter intervals where possible too.
- we only want to backup from linux to linux
- we only backup servers, so we can assume that both the client as well as the server are online 24/7, so scheduling by simple cron jobs or similar is enough. however a failed backup should be resumable at the next time the cron job runs again.
- backups should be incremental, so only files that where changed or new files since the last run should be submitted to the backup server. we do tolerate re-uploading an entire file, even if only a partial change was made..
- there should be a way to mount or at least view any backup as if it was a full backup. Most probably this will be achieved once again through hard-linking unchanged files
- the tool should be as simple to install as possible.. either by just copying a simple script and installing some very common linux tools like rsync etc. or by making it available as a docker container or similar
- we accept the fact, that there will most probably be a client and a server side script that needs to be put in place, running it all from one side won't be possible due to the security concerns above

discussion of available tools and solutions

i have discussed some tools mainly regarding encrypted backups already in [encrypted_backups_to_the_cloud](#). In addition to that i have looked at some other tools for this project:

Borg

[Borg](#) is a very smart and capable backup solution with lots and lots of features and most importantly, block level deduplication. It can do a lot more than what I need, **BUT** it sadly does not allow to create a user that can only write new backups but not delete old ones as well. While there is a append-only

policy available, it is impractical to rely on it, as one has to either forget about automatic purging of old backups. That's the deal breaker for me. Read more about this in the [FAQ](#) and the [Drawbacks of append-only mode](#)

Burp

[Burp](#) is the best backup tool i know to make backups of clients that aren't available 24/7 such as workstations or even notebooks. it also supports linux, windows and Mac OS. Burp actually does pretty much anything I want and a lot more out of the box. the only draw back that i found so far is, that it seems rather difficult to create secondary backups. there is an offsite-backup script but it says in the header of the script, that it doesn't quite work.. the main issue seems to be the fact, that burp moves the full-backup always along to the youngest backup and only keeps the previous versions of changed or deleted files stored in the previous backup's data directory. this makes it very easy to purge old backups of course, but it is a little trickier to create secondary backups.. i haven't tested if this can be overcome by using hardlinked backups (a config option in burp) though. burp also supports client side encryption which breaks delta uploads of modified files. At the end of the day, burp provides no real advantage to me over using a combination of some other tools and some custom scripts to glue them together, but it adds complexity mainly to the offsite backup part instead and it adds potential security risks by setting wrong configuration options. Still, it is a complete and running software and would probably save some time on my end, and it offers so many more features which i currently don't use (like windows backups) but that might come in handy in the future. So maybe my final solution could be writing a offsite-backup for burp to complete the required feature set for me :)

Restic

[Restic](#) seems to be an awesome tool that does almost everything i want my new backup tool to do.. Most importantly, it creates client-side encrypted incremental backups of your servers and can then store it to a broad range of storages available including S3 compatible storages etc. This is all very nice, **BUT** it is run on the client side only, which means, if a hacker gains control over your server and decides to encrypt or delete your data, he can simply delete all your backups and you are screwed.. so it sadly fails our security requirements.. but maybe it could be used as the client side of my backup solution, with a server that prevents deleting old backups from the client.. this will need some further research

possible solutions

Burp

Since burp has all the features we want, it seems like it might be worth to just dig into the remote-backup issue and fix that in whatever way i can find.

possible solutions to consider are:

- use burp in hardlink mode and try if using rsync is simpler now
- use burp with a BTRFS or ZFS storage underneath and then create a custom off-site backup (by

using BTRFS Snapshots).

- BTRFS seems to be a good choice as it is designed to provide easily transferrable snapshots.. by backing up each snapshot to the offsite server, we could make sure, that even if the entire btrfs filesystem is messed up on the main backup server, we would still have intact old snapshots on the secondary backup server.
- ZFS on the other hand provides erasure coding through the various raidz2 and so on levels, which means we will lose less harddisk space and can work without expensive and error prone raid controllers in our backup systems, that would be a nice side-effect
- the advantage of any of these snapshot method is, that they will work no matter what the burp developers decide to change in the future about how they store backups.. so we don't need to mess with internals of burp to get a remote backup running, where with rsync we need to know what burp does exactly and how we can back up the right data to preserve this functionality.

self-made collection of other tools

so "self made" is a bit flexible here.. what i mean is a larger script that will use a combination of several tools together:

- rsync as the main tool to copy data from the client to the server
- [GoCryptfs](#) in reverse mode on the client to provide client side encryption
- use rsync daemon on the server to provide access to the backup repos for each of the clients.
- use rsync `--link-dest` or `cp -alx` to first create a fully hard-linked copy to the last successful backup and then share this via rsync daemon for the client to then update the changed file in this repo.. this should probably result in a similar backup structure as my current moby script does, but with the added separation of client and server.
- provide a read-only share via rsync daemon where the client can access all its backups to restore files from. -> **this needs some more thinking / research**, as the backups will contain encrypted file- and directory names as well as data.. so we would need some other means of sharing the backups in read-only mode but that will retain the original linux permissions upon restore. the share should be mountable on the client, so that we can use again gocryptfs to decrypt the backup before restoring files. Maybe NFS piped through ssh or something similar might be a solution.
- use the same set of tools again to create backups from the primary backup server to the secondary.

Unsolved issues of this solution:

- **file ownership** is retained on all the files, so a file belonging to root on the client will belong to root on the backup server.. this brings some security issues, as for example a privilege escalation could be made possible by backing up a copy of bash belonging to root and with the suid bit set.. once the attacker gets unprivileged user access to the backup server, he could start this shell and become root. So it would be preferable to change at least file ownership to a dedicated user and limit the possibilities for an attack
- **restoring files and browsing backups** needs to be simple. for example it should be possible to either use normal rsync `-l` or even better, to mount complete backups from the backup server onto the client server and then browse through them. however, this is currently not so simple because:
 - backups are encrypted before rsync lays a hand on the file, so rsync `-l` will list encrypted file- and directory names and it will download encrypted files which will then

need to be decrypted.. so finding the latest version of a file that contains a string X for example is very cumbersome

- it would be nice to be able to mount an entire backup, or even all backups at once, via for example sshfs. One could then remount it using gocryptfs on the client to see a decrypted representation. however, this brings another issue: the mount should be read-only, so that a hacked client can't destroy existing backups on the backup server. so either we find a way to create a read-only share using for example NFS (possibly tunnelled over ssh) or we find a way to make them read-only on the backup server already before sharing them through sshfs.
- i have found [fuse-rsync](#) which allows mounting an rsync module via a fuse mount. however, this is merely a proof of concept that has not been developed any further in the past 7 years, so not really an option here.

From:

<http://wiki.psuter.ch/> - **pswiki**

Permanent link:

http://wiki.psuter.ch/doku.php?id=mobi_2.0&rev=1609401522

Last update: **31.12.2020 08:58**

