Rethinking my Backup Strategy

in this wiki you can find my current backup script mobi_backup which basically does everything i needed so far. However, after years of using it both privately as well as on customer installations, mostly for local backups, it is time to take a step back and re-think my backup strategy

The main reason why I feel my current backup solution is no longer the perfect solution for me 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 accidential 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 vincinity (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.

What i want from my new backup solution

- 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 form 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 versioins of changed or deleted files stored in the previous backup's data directory. this makes it very easy do 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. Burp is a complete and running software and would probably save some time on my end versus a home-made solution made of some other tools glued together, 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-bakckup 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** Restic itself runs on the client side only. This means, if you just use it like a standalone solution, an attacker who has (root)shell access to your system might delete your backups and then encrypt your data. The keyword here is “append only” backups and it must be implemented on the storage side. This in turn means, that restic won't be able to rotate backups anymore, so that needs to be implemented on the storage side. Restic comes with its own solution. They provide a REST server which has the option to accept append-only backups and therefore secures your backup from other attackers. This last part was something i had missed at first but with this server in the pbackage, it makes for a good alternative to burp.

dar

dar is a tool i have to take a closer look at.. it is used in various other backup software as an alternative to rsync and provides backup images which means we can retain permissions in backups even with only unprivileged access to the target storage. this would allow to implement a backup script that creates a backup to a write-only share which is then made available read-only for future use, hence protecting it from a possible attacker with control over the backup target. the big question is, how do they handle incremental backups with regards to rotating old backups and restoring backups. there is a decremental backup method available too (where the full backup is always the latest and then decrements (file states before they where changed to the current state) are saved for older backups). decrementals obviously create a lot of load at the end of a backup and will probably not work when older backups are read-only.. so this might be tricky.

SFTPgo

SFTPgo would make a good server side for a backup solution. Basically it comes with lots of features that can be helpful to implement an append only server to store backups. It is, as the name says, an SFTP server for linux and some more OS's which at first sounds strange, as linux already has openssh

which is broadly used for sftp as well because it is on almost every linux server out there to provide ssh access anyway. However, SFTPgo is much more than just a simple sftp server. it comes with its own user management, it provides more protocols such as webdav and it even accepts rsync connections. it can share more than just a plain linux filesystem too.. it can share S3 storage, provide DAR encryption and it supports setting very granular access rights on top of the posix layer. So with this a share could be configered where one would only have write but not overwrite or delete access. pretty nifty.

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 radz2 and so on levels, which means we will loose 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.

the main advantages of using burp over a selve made variant are

- burp is tested and trusted by many
- burp already solves the permission issues
- burp provides a way to browse client-side encrypted backups already
- we only have to implement a secondary backup solution. so this is probably a lot less work over all
- we get additional features such as windows backups for free which might be appreciated in the future.

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 encrpyted 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 orignal 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. shiftfs in combination with `unshare` to create a linux user namespace could be a solution here.
- **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 cumbresome
  - 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 isse: 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.

First POC - Burp + rsync

with all the arguments above considered, I decided to proceed a burp based solution and just add off-site capabilities to burp. Here is the targeted setup:

- “Local” backup server running burp in server mode with the following key settings:
  - `hardlinked_archive = 1`
  - `client_can_delete = 0`
  - `user=jdoe` and `group=jdoe` where `jdoe` is some unprivileged non-root user
  - one needs to make sure that all the necessary paths mentioned in `burp-server.conf` and `CA.cnf` are writable and or readable by the unprivileged user who's running burp
- “Remote” backup server, running a rsyncd service which shares a single directory i.e. `/backups/current`
- clients run the burp client and use client-side encryption with a strong password. the following additional core settings are used:
  - `server_can_restore = 0`
  - `server_can_override_includes = 0`
- a script on the burp server uses `rsync -aAhHvXxR –numeric-ids –delete …` sync the latest backups stored in `/var/spool/burp/<client name>/<backup timestamp>` where `backup timestamp` is the target of the `current` symlink in the same directory. so a script which uses something along the lines of `readlink` should find the `current` backup which was just completed and then rsync this to the offsite server's `current directory`
- on the offsite server, a script is called (somehow, haven't figured out yet how exactly this will be done) after the rsync from the burp server successfully finished. the script will use `cp -alx /backups/current /backups/`date +%Y.%m.%d-%H%M`` to create hardlinked copies of the current directory. by using this script, we can avoid to use the `--link-dest` option of rsync which in turn would make it necessary to at least include the latest completed backup also in the writable share.

to try it all out i used a bunch of ubuntu test docker containers

```
docker network create burp
```

to create the custom network

```
docker run --net burp --name burpsrv -ti ubuntu-test:latest
```

to create the container for the burp server, and similar commands for the other servers.

for testing i ran the burp server with this command line:

```
burp -v -F -c /home/jdoe/burp/etc/burp-server.conf
```

which outputs any logs directly to stdout and keeps the daemon in the foreground.

offsite backup file encryption

since all files are encrypted on the client side before they are sent to the backup server, we don't have to encrypt them again when uploading them from the backup server to the offsite backup. this basically removes all the challenges mentioned in the “self made” backup solution above.

restore from an offsite backup

here is how i have tested the restore-ability of an offsite backup in case we have completely lost the backup server in between.

1. set up a new burp backup server with the same client config
2. setup a new client or delete the certificates if the client is still there and should be re-used
3. using `burp -a l` execute the initial connection between the client and server and let burp create all the SSL keys.
4. using `rsync -aAHhvXxR –numeric-ids` copy the desired backup to the `/var/spool/burp/<client name>/<backup name>` folder where `backup name` is identical to the one stored on the offsite server.
5. now use `burp -a l` again and you should see the backup listed
6. restore using `burp -a r -b 5 -d /` or a similar command, depending on your situation to restore the specified backup

this worked flawlessyl in my test, of coures as long as I still had the **encryption password** available from somewhere! .. needless to say, if you don't store your encryption password your backup is completely useless, so make sure your encryption password is saved somewhere where you will still have it, even if you loose the client, and of course, don't store it together with the backup on the backup servers :)

From:
http://wiki.psuter.ch/ - **pswiki**

Permanent link:
http://wiki.psuter.ch/doku.php?id=rethinking_my_backup_strategy&rev=1628782186

Last update: **12.08.2021 17:29**