# My new Backup solution with Burp

I have long used an rsync wrapper script I had written which collected backups from all of my servers and stored them locally, then another rsync wrapper was used to rsync the latest backup to an offsite location.

While this solution works great against accidental data-loss, it can be pretty useless and even an additional risk when it comes to data loss due to malicious software or hackers. As a matter of fact, with my old system, the backup server needed access to all the files it should back up, in other words, if an attacker (person or software) gained access to the backup server, they would have automatically also gained access to all the data that was being backed up.

my new solution is now based on Burp, a solid and proven open source **B**ack**U**p and **R**estore **P**rogram :) with the right configuration, Burp can operate so that a hacker could neither destroy the original data after taking control over the backup server, nor could the attacker render the backup useless after hacking one of the backed-up machines. I took it a step further and added an offsite backup which gain stores multiple versions of the original backups and also there, gaining control over the remote backup server does not allow to delete original backups and gaining control over the backup server does not allow to delete offsite backups.

# Server preparation

the following setup was done on an ubuntu server, similar things need to be considered on other distributions.

in order to only expose what's necessary, it is always a good idea to enable some kind of firewall on the server. In my case i use “ufw” which is very simple to use:

allow ssh access

```
ufw add ssh
```

allow burp server access

```
ufw add 4971/tcp
```

this is needed for our offsite-backup sftp server

```
ufw add 2022
```

finally enable the firewall

```
ufw enable
```

install burp:

```
apt update
apt install burp
```

add the user to run burp with:

```
useradd -r burp
nano /etc/passwd
```

change login shell to `/usr/sbin/nologin`

fix permissions of config files and dirs

```
chown -R burp.burp /backup /etc/burp
touch /var/run/burp.server.pid
chown burp.burp /var/run/burp.server.pid
nano /etc/burp/burp-server.conf
```

of course mount your backup target storage

# Burp configuration

In order to achieve the above mentioned security level, one needs to set a few config options correctly, both on the burp server as well as on the client

## Server Side

in `burp-server.conf` make sure the follwing settings are as listed here:

```
hardlinked_archive = 1
```

this minimizes post-backup work on the backup server and it is necessary for my offsite-backup solution

```
client_can_delete = 0
```

without this option, an attacker could delete all backups of a host after taking control over the host itself. With this option enabled, the attacker can only see that there are backups, but they can't delete them.

```
user=burp
group=burp
```

user to run as, otherwise burp runs as root which is an unecessary security risk

```
client_can_force_backup = 0
```

without this option, a hacker could force enough backups to kick all valid old backups out of retention after destroying you data. so you would be left with a backup server full of nonesense. Worst case an attacker could change the client config to only backup an almost empty directory, in which case a backup is done very quickly. like that, it doesnt' take long to create enough new backups that all your old ones are destroyed by the backup rotation mechanism. If this option is set to 0 however, a backup can only start if it is due according to the schedule which is defined on the server.

```
compression = zlib0
```

nowadays, storage is way too cheap to use zlib compression. per default, burp compresses all your data on the backup server to save space. the problem with this is, that if you have a lot of data, it takes extremely long to do that, and if you only have little data, it doesn't really save you much.

to configure new clients, a new config file needs to be created for each client in the `clientconfdir` directory. in these files, you should define at least a safe password to authenticate the client. you could use pwgen to do that in a one-liner such as:

```
echo "password = $(pwgen -s 35 1)" | tee /etc/burp/clientconfdir/newclient1
```

this password is needed for the client to be configured. once this is done, you won't have to type or use it ever gain unless you need to restore an entire client machine after a catastrophic failure or hardware change.

## client side config

on the burp client (the machine that is being backed up) there are also a few imortant settings:

```
encryption_password
```

set this to a very long and safe password and **note the password down**. Store it in a place where you can find it again should you ever need to perform a full restore of a server. This password is not stored on the server, so if you lose it and your burp client is gone, your backup is completely useless as you can't restore it! So really, make sure you don't lose this password but at the same time **don't save it on the server or anywhere lese where you backup data may reside**. If you stored this on the server, an attacker with control over your backup server could restore a backup of another server and eventually extract security credentials or other data that could help them to gain access to the other machine as well.

```
server_can_restore=0
```

with this set to 1, an attacker could potentially push data to the client by forcing a restore of a tampered backup or they could simply erase data by restoring an old version of certain files. we don't want that!

## first start

start burp on the backup server

```
systemctl enable --now burp
```

in case it won't start and shows an error with the CA generation,stop the service, then delete

```
/etc/burp/CA
/etc/burp/CA-client
/etc/burp/ssl*
```

and try again.

try to connect from the backup client

```
burp -a l -v
```

if that was successful, add a cron job that runs burp every 20 minutes or so. this does not mean that your data is being backed up every 20 minutes, it means that the burp client will ask the burp server every 20 minutes if it is time to start a new backup or continue with a halted one. the actual schedule is defined on the server side (in the server config or clientconf files).

```
0,20,40 * * * * /usr/sbin/burp -a t -q 1200 >>/var/log/burp-client 2>&1
```

## setup a backup report script

i wrote a script to send reports on a daily basis listing all backups and their age. since burp does not run a backup of all machines at once, but rather lets the client machines start a new backup whenever it is due, we have to create a backup report independent of any running jobs at a fixed time of the day (or week). I personally like systems that also send an email when things are good. this assures that you will get the alert if one is needed.

## offsite backup solution

theoretically one could probably use burp to create an offsite backup of another burp server. However, i wanted something different, so in case there is a security issue with burp, the offsite backup would not automatically be compromised as well.

I chose to use a combination of rclone on the offsite machine and sftpgo on the burp server. Basically sftpgo is a dedicated sftp server daemon. It does not have the full ssh functionality such as port forwarding, shell etc, it only allows file access. It also supports a few other protocols, but we will stick to sftp for now. Another advantage over just configuring a very limited user for our normal OpenSSH server is, that it sftpgo can also create read-only file shares, which is exactly what we want.

So set up sftpgo and then configure a new user (which should not be a sytem user, it's an internal user of sftpgo). Then share the backup directorie(s) with read-only access to this user.

now depending on your offsite backup machine, if it has a fixed ip address, you can create a more specific rule for ufw, that allows only access from this specific IP. This is not possible if your remote server has a dynamic ip of course.

```
ufw allow proto tcp from 172.16.28.1 to any port 2022
```

my offsite backup wrapper script for rclone will be able to work on multiple offsite backups at the same time, so make sure you allow enough concurrent connections on the sftpgo site. The default of 20 was not enough for 4 concurrent rclone downloads, i had to set it to something above 30. went to 100 and haven't had an issue since:

```
sed -i 's/\("max_per_host_connections":\) .*/\1 100/'
/etc/sftpgo/sftpgo.json
systemctl restart sftpgo.service
```

# Offsite-Backup server

on the offsite backup server, isntall the latest version of rclone

then run

```
rclone config
```

to set up a new client.

add the backup host and make sure that you pass the path to the unencrypted ssh key where it asks for the `ssh_key` and not the `ssh_pem` .. it is NOT the Raw `pem-encoded` key, it is just the `pem-encoded` key which was confusing to me :)

```
rclone lsd planb:/
```

should now show our backups

Now download my rclone wrapper to create offsite backups of a burp server and set up a cron job for it. (currently this script is a WIP, so come back later when it's done)

## Disaster Recovery fron the offsite backup

this is the process after we have lost it all.. say a fire has hit our facility and everything is gone except for our offsite backup or maybe even only the tape has survived

## Restore the "burp" backup server

we assume that we have a backup of our former burp server and that the backup name is “backupserver”

### 1.) setup temporary burp server

for this tutorial we will use an ubuntu docker container as our temporary burp server to restore the actual backup server from it.

first from the offsite or tape backup, copy the `backupserver` subfolder to a temporary directory on the machine that is going to run the docker container. let's assume it is saved to `~/tmp/burp_restore/backup/backupserver`

start the docker container:

```
docker run -ti --rm -p 4971:4971 --name burp-server -v
~/tmp/burp_restore//backup:/var/spool/burp ubuntu:latest
```

this will open a shell inside the container. In this shell, we will install burp and get the burp server configured and running:

```
apt update && apt install burp
sed -i 's/hardlinked_archive.*$/hardlinked_archive=1/' /etc/burp/burp-
server.conf
echo "password = abcdefgh" > /etc/burp/clientconfdir/backupserver
```

no joke, the password set here is the default in the sample client config, because we are lazy and this is a temporary server only, we will set it on the server side to match the client so we don't need to adjust it on the clinet later on :)

start burp server

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

### restore "backupserver" server onto future burp server

boot future “backupserver” burp backup server from an ubuntu desktop live stick. choose to `try ubuntu` rather than install it.

make sure it is connected to the same network as our burp docker container above.

open a shell and become root

```
sudo su -
```

install burp

```
apt update
add-apt-repository universe
apt insatll burp
```

temp0raryPW

```
sed -i 's/^server = .*$/server = <ip of machine running the docker
container>/' /etc/burp/burp.conf
sed -i 's/^cname = .*$/cname = backupserver/' /etc/burp/burp.conf
sed -i 's/.*encryption_password =.*$/encryption_password =
<encryptionPassword>/' /etc/burp/burp.conf
```

replace the “<encryptionPassword>” with the actual encryption password which you got to have saved somehwere, otherwise you can't restore this server anymore. if you have the encryption passwords for your other servers, follow this tutorial for each of them and you should be able to restore them at least.

run burp client to list the available backup. amongst all the lines of output there should be a line that start with `Backup: ….` which shows the backup number which we want to restore.

```
burp -a l | grep "^Backup:"
```

this should sow something like

```
Backup: 0000002 2021-08-09 06:04:07 +0200 (deletable)
```

where 0000002 is the number of the backup.

now restore `/etc/fstab` to find out what the partitioning of the new OS disk should look like if you are uncertain what the partitioning was. `fstab` won't tell you the partition sizes, use your own judgement for that, but it will give you an idea what partitions where mounted where.

```
burp -a r -b 2 -r "^/etc/fstab$" -d /tmp/
```

NOTE, if the backup server is still a legacy booted machine, so there is no UEFI partition to be made, instead craete a 1MB “BIOS Boot” type id 4 partition at the begining of the partition table, then simply a root partition and a swap partition, that's it.

now format the root partition with ext4

mount the root partitioin to `/mnt/` in your live linux session.

and now restore the backup to the new disk:

```
burp -a r -b 2 -d /mnt/
```

now chroot into the restored disk to setup the rest:

```
for d in dev sys proc ; do mount -o bind /$d /mnt/$d; done
chroot /mnt/
```

use mkswap to format the swap partition

```
mkswap /dev/<target>
```

update the UUID's in fstab. for example, dump the uuid's into fstab, then edit the file and move them to the right place:

```
blkid >> /etc/fstab
nano /etc/fstab
```

now install grub

```
grub-install
grub-mkconfig
```

finally reboot and boot from the harddisk

```
exit
umount /mnt/*
umount /mnt
reboot
```

finally, stop the temporary burp server docker image, just press ctrl+c and then type exit.