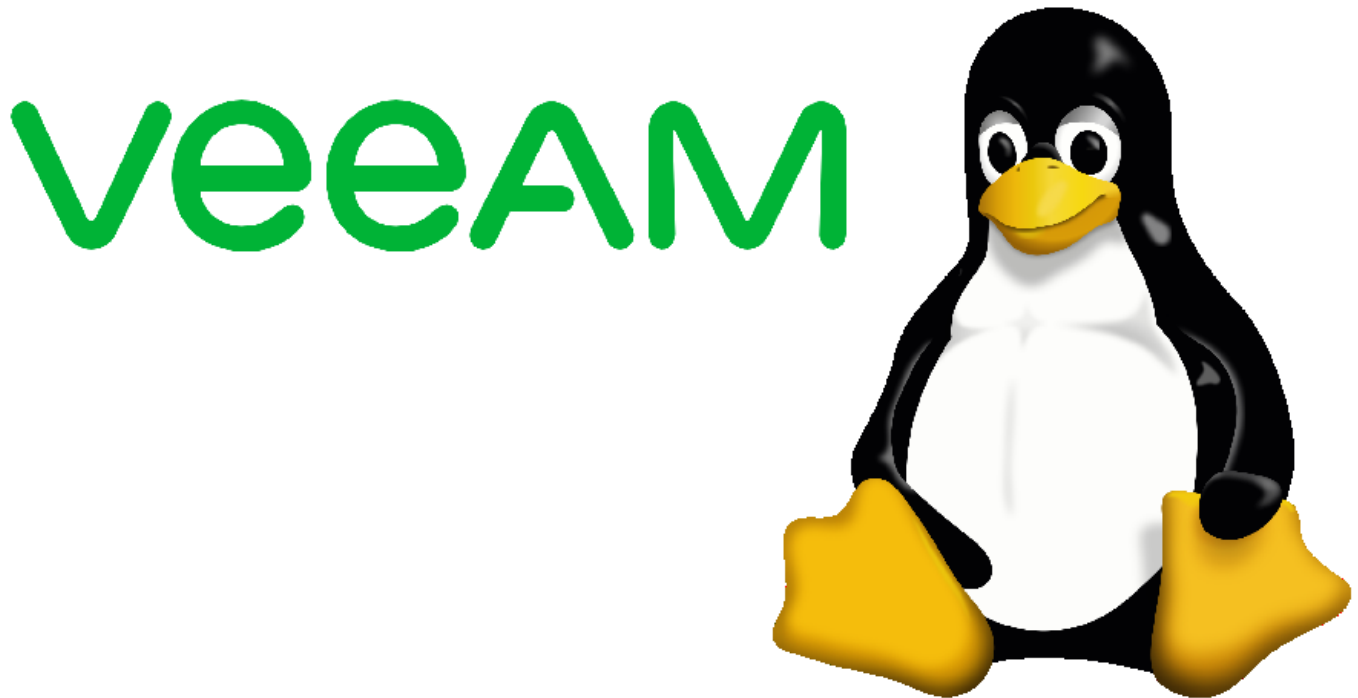


Build an immutable backup repository for Veeam Backup & Replication. Part 8



This guide will show you, step by step, how to create and implement a disk-based immutable Veeam backup repository from scratch. In this part: Tighten security on the Linux server (MFA/2FA).

Introduction

Purpose of these articles

You are a Windows administrator running [Veeam Backup & Replication](#) and wish to raise protection against malware attacks and hackers without reverting to shuffle or rotate physical media.

This you can accomplish by *immutable backups* stored on a physical server running Linux. However, you have no Linux servers running and don't want to.

But, like it or not, that is your only option, as the XFS file system is the only one capable of immutability, and XFS only runs under Linux.

Thus, a Linux server is a must. When you have accepted this fact, then what? Where to start?

Like me, you have about zero experience with Linux and, therefore, hesitate to set up a Linux server, indeed in a production environment.

If so, this guide is for you. Here, nothing about Linux is taken for granted.

Sections

The guide has been split in eight parts. This allows you to skip parts you are either familiar with or wish to implement later if at all.

1. [Prepare the install of Linux](#)
2. [Install Linux on the server](#)
3. [Prepare the Linux server for Veeam](#)
4. [Create the immutable Veeam backup repository](#)
5. [Prepare for backup of the Linux server itself](#)
6. [Backup of the Linux server itself](#)
7. [Bare Metal Recovery of the Linux server](#)
8. [Tighten security on the Linux server \(MFA/2FA\)](#)
9. [Maintenance and deactivation/reactivation of MFA/2FA](#)

Requirements

You are familiar with:

- the usual tasks administering at least a small network with one Windows Server
- *Veeam Backup & Replication* and have it installed and running
- the command line - from PowerShell, Command Prompt, or even DOS

Veeam Backup & Replication is assumed to be of *version 11* or later. It can be a licensed trial or paid version or even the free [Community Edition](#).

Part 8. Tighten security on the Linux server

In the previous sections, we equipped the Linux server with a proven backup system to be able to quickly perform a *bare metal recovery* and bring the server online, in case the system drive should cease to function.

In this section some methods will be shown to tighten the security on the server. All of these are related to SSH, the method we use to remote control the server as it, in many cases, is quite cumbersome to be forced to sit and physically operate the server via a directly connected keyboard and monitor.

SSH is, in general, considered to be extremely secure and, by all practical measures, impossible to crack. However, as they say, "you never know",

The ultimate method is, of course, simply to shut down the SSH server on the Linux server. However, a less brute and more flexible method is to *deactivate* the service.

Deactivate SSH on the Linux server

One safe method to disable remote access to the Linux server is, of course, to shut down the SSH service itself. You don't have to uninstall the service, only stop it.

To view the status of SSH, run this command:

```
sudo systemctl status sshd.service
```

If running normally, you will see the current status and the recent log:

```
linuxadmin@vubuntum: ~  
linuxadmin@vubuntum:~$ sudo systemctl status sshd.service  
● ssh.service - OpenBSD Secure Shell server  
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)  
   Active: active (running) since Sat 2022-01-01 18:23:57 CET; 1min 54s ago  
     Docs: man:sshd(8)  
           man:sshd_config(5)  
  Process: 1667 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)  
 Main PID: 1668 (sshd)  
    Tasks: 1 (limit: 2200)  
   Memory: 3.4M  
    CGroup: /system.slice/ssh.service  
           └─1668 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups  
  
Jan 01 18:23:57 vubuntum systemd[1]: Starting OpenBSD Secure Shell server...  
Jan 01 18:23:57 vubuntum sshd[1668]: Server listening on 0.0.0.0 port 22.  
Jan 01 18:23:57 vubuntum sshd[1668]: Server listening on :: port 22.  
Jan 01 18:23:57 vubuntum systemd[1]: Started OpenBSD Secure Shell server.  
linuxadmin@vubuntum:~$
```

Notice, that *active (running)* is green.

To stop the SSH service, run this command:

```
sudo systemctl stop sshd.service
```

Note, that this command does not close a current connection; you can call this command from an SSH client, view the status, and even restart the service, if you like.

Checking the status, when the service is stopped, will show this:

```
linuxadmin@vubuntum: ~  
linuxadmin@vubuntum:~$ sudo systemctl status sshd.service  
● ssh.service - OpenBSD Secure Shell server  
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)  
   Active: inactive (dead) since Sat 2022-01-01 18:27:54 CET; 49s ago  
     Docs: man:sshd(8)  
           man:sshd_config(5)  
  Process: 1801 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)  
  Process: 1817 ExecStart=/usr/sbin/sshd -D $SSH_OPTS (code=exited, status=0/SUCCESS)  
 Main PID: 1817 (code=exited, status=0/SUCCESS)  
  
Jan 01 18:27:47 vubuntum systemd[1]: Starting OpenBSD Secure Shell server...  
Jan 01 18:27:47 vubuntum sshd[1817]: Server listening on 0.0.0.0 port 22.  
Jan 01 18:27:47 vubuntum sshd[1817]: Server listening on :: port 22.  
Jan 01 18:27:47 vubuntum systemd[1]: Started OpenBSD Secure Shell server.  
Jan 01 18:27:54 vubuntum systemd[1]: Stopping OpenBSD Secure Shell server...  
Jan 01 18:27:54 vubuntum sshd[1817]: Received signal 15; terminating.  
Jan 01 18:27:54 vubuntum systemd[1]: ssh.service: Succeeded.  
Jan 01 18:27:54 vubuntum systemd[1]: Stopped OpenBSD Secure Shell server.  
linuxadmin@vubuntum:~$
```

Notice, that the status now is *inactive (dead)* and neutral colour.

After a logout in the SSH client, you can't login again. If you try, you'll see that the connection is refused:

```
Windows PowerShell  
PS C:\Users\Cactus.CACTUS> ssh linuxadmin@192.168.1.26  
ssh: connect to host 192.168.1.26 port 22: Connection refused  
PS C:\Users\Cactus.CACTUS>
```

To restart the SSH service, run this command:

```
sudo systemctl restart sshd.service
```

Remote connection is now again possible.

Scenario for operation with closed SSH

If you prefer normally to have no remote access to the Linux server, a procedure to allow for remote control for a single session could be:

- go to the server, login via physical keyboard and monitor, activate SSH
- go to your workstation, connect to the Linux server with your SSH client
- do your work via your SSH client
- via your SSH client, deactivate SSH on the server, and logout from the session

Remote control via SSH has again been deactivated.

Firewall

If the above method is too restrictive, a permanent solution could be to move the Linux server behind a separate firewall.

A few ports and port ranges need to be open for the TCP protocol for *Veeam Backup & Replication* to be able to control the repository hosted by the Linux server:

Port	Notes
22	Port used as a control channel from the console to the target Linux host
6162	Default port used by the Veeam Data Mover
2500 to 3300	Default range of ports used as data transmission channels. For every TCP connection that a job uses, one port from this range is assigned.

Full documentation on ports used by Veeam can be found here:

[Linux server connections](#)

In the firewall, you could set up rules that block connections to SSH from all IP addresses except those from your Veeam server and that (those) of your workstation(s). This will force a hacker or the malware to first gain access to either of these machines, or to know an IP address of these and how to spoof it. This is doable, of course, but difficult.

Another option is to have admin control to the firewall and have the SSH connection option from your workstation to the Linux server normally blocked. Then, to remote control the Linux server:

- login to the firewall and enable the SSH rule for your workstation to connect
- login to the Linux server and do your work
- logout from the Linux server
- login to the firewall and disable the SSH rule for your workstation

Alternatively, you could open a *Remote Desktop* connection to the Windows server hosting *Veeam Backup & Replication*, and then use PowerShell on the server to connect to the Linux Server.

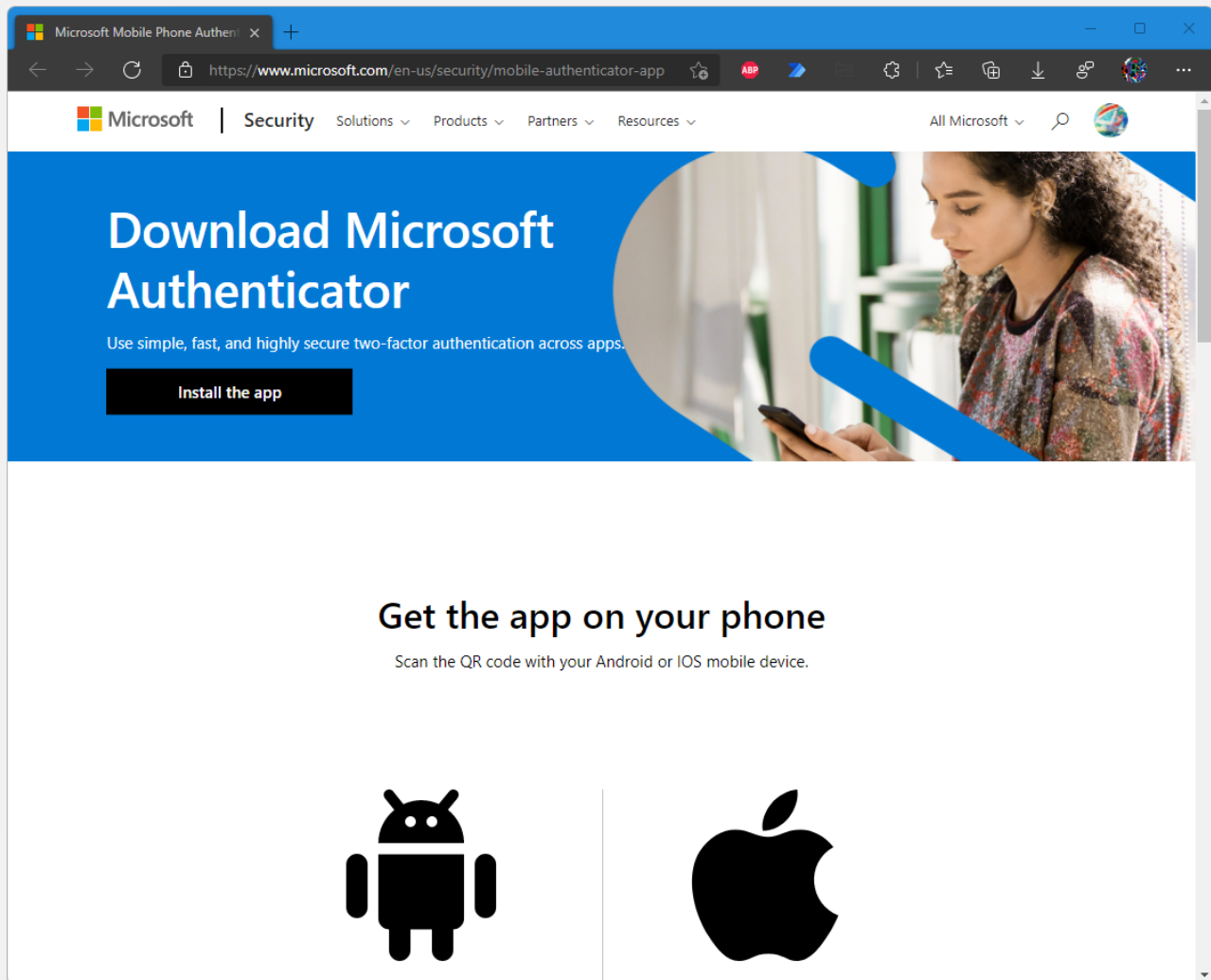
Implement MFA/2FA authentication

Finally, as you probably already have implemented for admin accounts in your Windows environment, you can install and force **MFA** (Multi Factor Authentication) or **2FA** (Two Factor Authentication) on the Linux server to login to this.

For this we will use the *Microsoft Authenticator* as you most likely already have this installed on your phone.

If you don't have the *Microsoft Authenticator* installed on your phone, get it now from here:

Microsoft Authenticator



Browse a page down to reach the large QR code. Scan this with the phone to install the app.

Install the Google Authenticator PAM

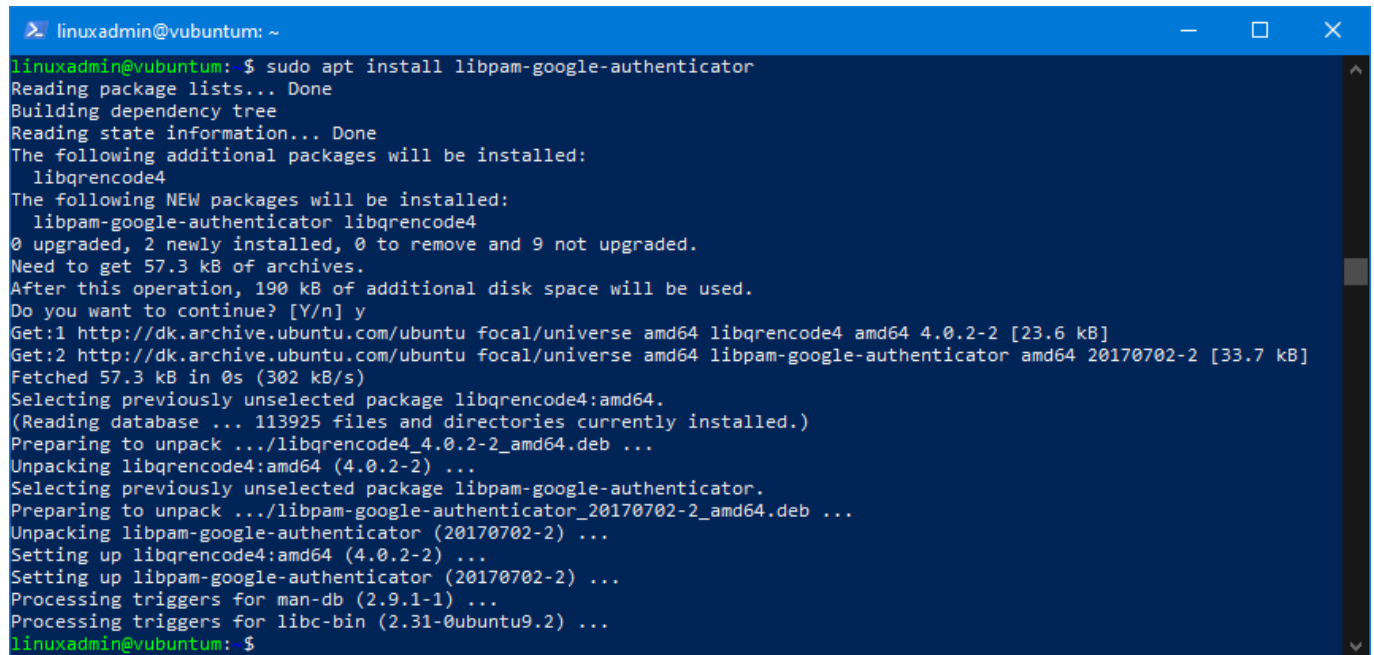
First, to enable 2FA on the server, you need to install a *Pluggable Authentication Module* (PAM), which offers the necessary infrastructure to integrate multi-factor authentication for SSH in Linux.

Google Authenticator PAM is the most popular PAM as it is quite easy to implement and use. It offers all the necessary functionality required to authenticate users using *Time-based One-Time Password* (TOTP) codes (six-digit codes).

To install *Google Authenticator PAM*, run the following command:

```
sudo apt install libpam-google-authenticator
```

Enter y at the installation prompt to confirm the process. It will go like this:

A terminal window titled 'linuxadmin@vubuntum: ~' showing the command 'sudo apt install libpam-google-authenticator'. The output shows the package lists being read, the dependency tree being built, and the state information being read. It lists additional packages to be installed: libqrencode4. The following NEW packages will be installed: libpam-google-authenticator libqrencode4. It shows 0 upgraded, 2 newly installed, 0 to remove and 9 not upgraded. It indicates a need to get 57.3 kB of archives and that after this operation, 190 kB of additional disk space will be used. It asks 'Do you want to continue? [Y/n] y'. It shows the download of libqrencode4:amd64 and libpam-google-authenticator:amd64. It shows the unpacking of the packages and the setting up of the packages. It shows the processing of triggers for man-db and libc-bin. The terminal ends with the prompt 'linuxadmin@vubuntum:~\$'.

Next, SSH must be tweaked a little to use the installed PAM for authentication. For this, you need to edit a couple of configuration files.

First configuration file

The first file is a lengthy file.

You can view the file in an editor using this command:

```
sudo nano /etc/pam.d/ssh
```

When done, exit the editor with *Ctrl+X*.

To insert the lines in the file, run these three commands:

```
echo '# Google Authenticator.' | sudo tee -a /etc/pam.d/ssh
echo 'auth required pam_google_authenticator.so' | sudo tee -a /etc/pam.d/ssh
echo '' | sudo tee -a /etc/pam.d/ssh
```

You can review the changes to the file in an editor using this command:

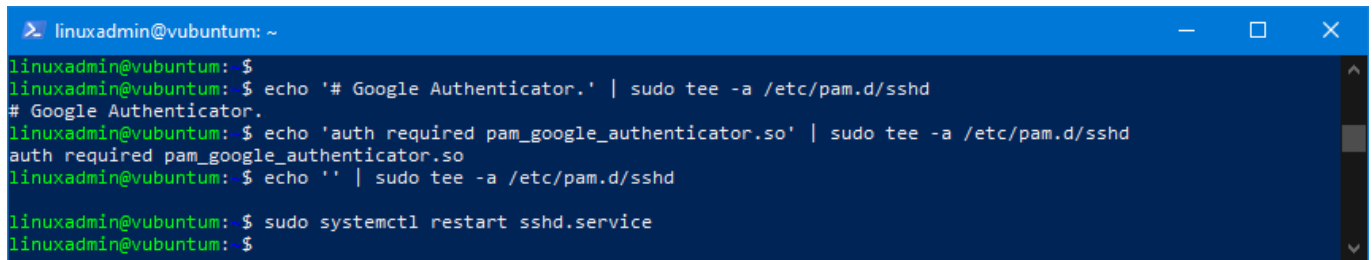
```
sudo nano /etc/pam.d/ssh
```

When done, exit the editor with *Ctrl+X*.

Then restart the SSH service with this command:

```
sudo systemctl restart sshd.service
```

The terminal window should now look like this:

A terminal window titled 'linuxadmin@vubuntum: ~' with a blue header bar. The terminal shows the following commands and output:

```
linuxadmin@vubuntum:~$  
linuxadmin@vubuntum:~$ echo '# Google Authenticator.' | sudo tee -a /etc/pam.d/sshd  
# Google Authenticator.  
linuxadmin@vubuntum:~$ echo 'auth required pam_google_authenticator.so' | sudo tee -a /etc/pam.d/sshd  
auth required pam_google_authenticator.so  
linuxadmin@vubuntum:~$ echo '' | sudo tee -a /etc/pam.d/sshd  
  
linuxadmin@vubuntum:~$ sudo systemctl restart sshd.service  
linuxadmin@vubuntum:~$
```

Second configuration file

Open the second file using *nano*:

```
sudo nano /etc/ssh/sshd_config
```

In this file, locate the line:

```
ChallengeResponseAuthentication no
```

and change its status from "no" to "yes". This will instruct SSH to ask for an authentication code whenever someone attempts to log in to the system.

Press *Ctrl+O* and *Enter* to save the file, then *Ctrl+X* to exit.

Once again, restart the SSH service:

```
sudo systemctl restart sshd.service
```

Link the server in the authenticator

First, make sure the *Microsoft Authenticator* is properly installed on your phone as described above.

If so, call this command in the terminal:

```
google-authenticator
```

and it will display a large QR code:

```
linuxadmin@vubuntum: ~  
linuxadmin@vubuntum:~$ google-authenticator  
Do you want authentication tokens to be time-based (y/n) y  
Warning: pasting the following URL into your browser exposes the OTP secret to Google:  
https://www.google.com/chart?chs=200x200&chld=M|0&cht=qr&chl=otpauth://totp/linuxadmin@vubuntum%3Fsecret%3DRLLB2SU5ER2  
OA76JIBHP5AXT74%26issuer%3Dvubuntum  
  
Your new secret key is: RLLB2SU5ER2OA76JIBHP5AXT74  
Your verification code is 935581  
Your emergency scratch codes are:  
18228377  
13519266  
82245609  
12570629  
52375885  
Do you want me to update your "/home/linuxadmin/.google_authenticator" file? (y/n)
```

Important: Following the QR are listed the literal key, code, and the *emergency scratch codes*. Copy these and save them in a safe place.

Pick your phone and:

- Open the Microsoft Authenticator
- Press the + sign at top right
- Select: Other (Google, Facebook, etc.) >

Then scan the QR picture. When captured, the app will add the server to the list and identify it by:

- hostname
- useraccount@hostname

as you can see here at the bottom:





Authenticator



ALSO Login
0010321868



597 995 



Contoso
jeffh@CRM529692.OnMicrosoft.c...



Simply.com
gustav@cactus.dk



274 539 



bastille
cactus@bastille



559 052 



vubuntum
linuxadmin@vubuntum



385 511 

000 0 14



Authenticator

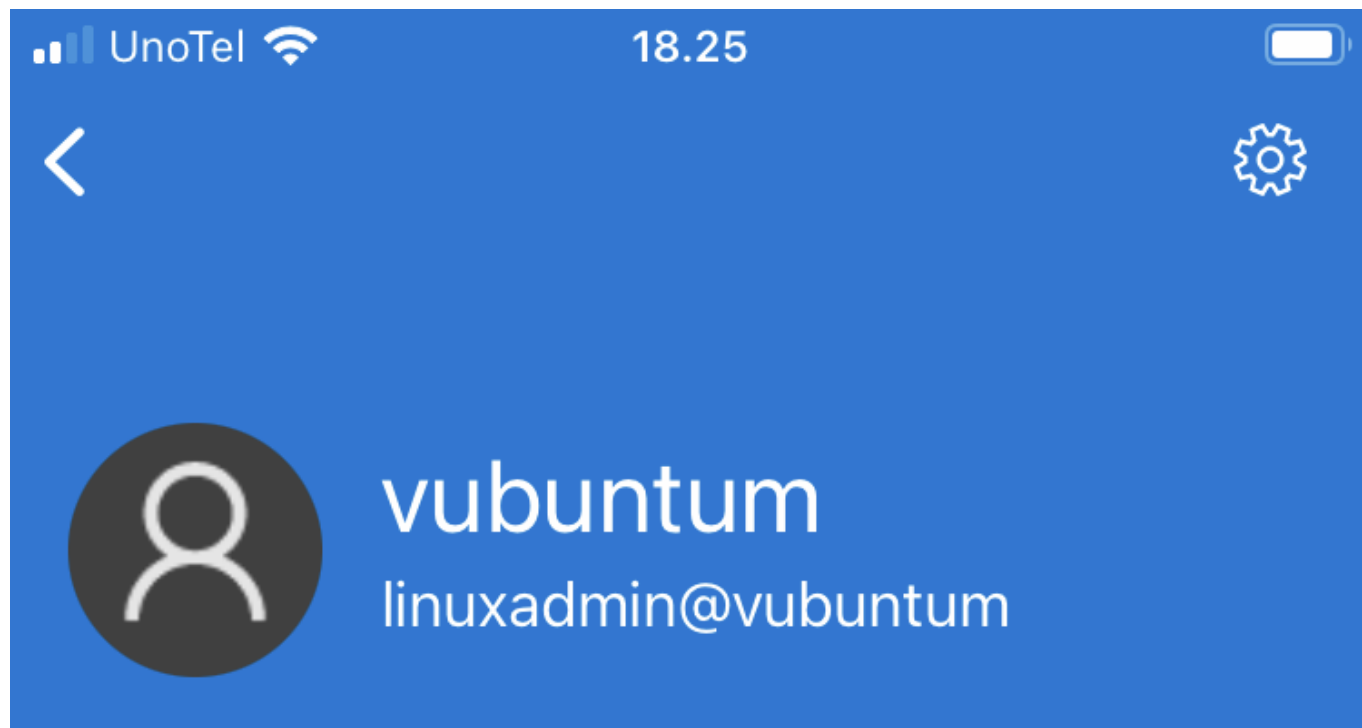


Adgangskoder



Adresser

Tap the small arrow (>) at the right to study the few details:



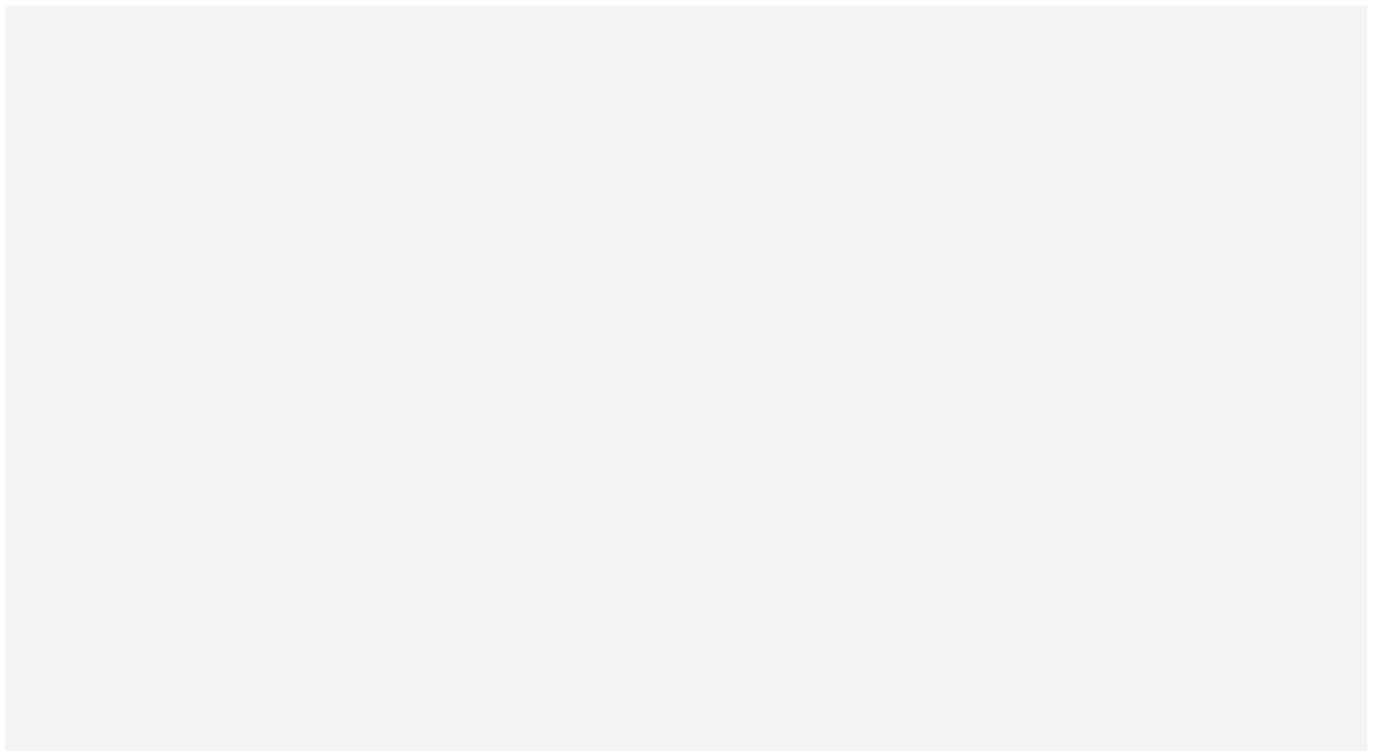
Engangsadgangskoder er aktiveret

Du kan bruge de engangsadgangskoder, der er oprettet af denne app, til at bekræfte dine logons



Engangsadgangskode

214 262



Put the phone away for a moment and return to the terminal window where the large QR still is on display.

Following the key and codes (see the image above with the QR code), you'll see a question.

Answer Y to this, and a series of questions will pop forward. Also answer Y to these to accept the default settings:

```
linuxadmin@vubuntum: ~  
Do you want me to update your "/home/linuxadmin/.google_authenticator" file? (y/n) y  
Do you want to disallow multiple uses of the same authentication  
token? This restricts you to one login about every 30s, but it increases  
your chances to notice or even prevent man-in-the-middle attacks (y/n) y  
By default, a new token is generated every 30 seconds by the mobile app.  
In order to compensate for possible time-skew between the client and the server,  
we allow an extra token before and after the current time. This allows for a  
time skew of up to 30 seconds between authentication server and client. If you  
experience problems with poor time synchronization, you can increase the window  
from its default size of 3 permitted codes (one previous code, the current  
code, the next code) to 17 permitted codes (the 8 previous codes, the current  
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes  
between client and server.  
Do you want to do so? (y/n) y  
If the computer that you are logging into isn't hardened against brute-force  
login attempts, you can enable rate-limiting for the authentication module.  
By default, this limits attackers to no more than 3 login attempts every 30s.  
Do you want to enable rate-limiting? (y/n) y  
linuxadmin@vubuntum:~$
```

That's it! You have now enabled 2-factor authentication for SSH connections on the Linux server.

Verify 2FA/MFA login

To confirm that 2FA/MFA is active, log out of the console and log in again.

As usual, connect with the user account and hostname/IP-address:

```
ssh useraccount@ip-address
```

and then it will ask for:

- Password:

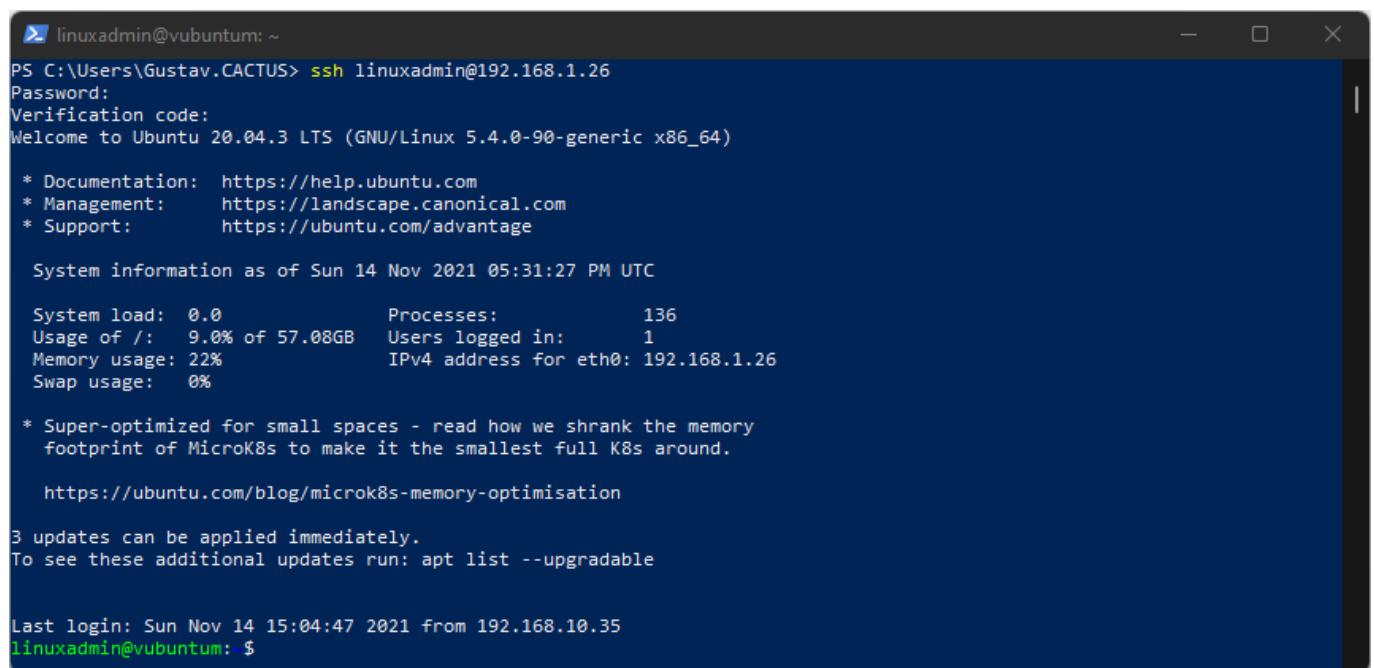
Enter this, and now it will also ask for:

- Verification code:

To obtain this code, follow these steps:

- pick your phone
- open *Microsoft Authenticator*
- browse to the entry for the Linux server
- read the displayed code
- enter the code *without the space*, like: 385514

The login will now proceed as usual:



```
linuxadmin@vubuntum: ~
PS C:\Users\Gustav.CACTUS> ssh linuxadmin@192.168.1.26
Password:
Verification code:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-90-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sun 14 Nov 2021 05:31:27 PM UTC

System load:  0.0               Processes:    136
Usage of /:   9.0% of 57.08GB   Users logged in: 1
Memory usage: 22%              IPv4 address for eth0: 192.168.1.26
Swap usage:  0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

3 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Last login: Sun Nov 14 15:04:47 2021 from 192.168.10.35
linuxadmin@vubuntum:~$
```

The 2-factor authentication is now implemented and tested, and a security layer has been added that is extremely difficult to break.

Conclusion

In this section, three additional steps to tighten security on the Linux server have been demonstrated. Each of these can be implemented either on its own or in a combination of two - or even all three - as to your preferences.

Epilogue

Keep in mind, though, that everything falls to the ground, if the bad guy (black hat) can get physical access to the Linux server.

This is the last part of the series of articles that deals with the installation of the immutable repository. The last covers a few maintenance tasks:

[Build an immutable backup repository for Veeam Backup & Replication. Part 9](#)