Configuring two-factor authentication in Ubuntu 14.04 using Google Authenticator

Configuring two-factor authentication in Ubuntu 14.04 using Google Authenticator

Category: Cloud Servers &nbsp
Access control is one of the more important aspects of running a server. Making sure that users only have access to what they need and not what they don't. More importantly, access control makes sure that those who shouldn't have access are kept out.

This is all the more important when restricting access to the shell, from where attackers can further compromise a system. In Linux systems with remote access this means securing SSH. By default, access to SSH can be controlled using a username and password combination or using the username with a public/private key pairing. While keys offer a massive security increase over passwords, and systems can be configured to block any access without a key, they are also quite limiting when it comes to gaining access as you are limited to only connecting from systems where the private key is stored. Also, if the system with the private key is compromised that can then be used to gain access.

Two-factor authentication increases your security by adding a second step to connecting to the server using different authentication factors for each step. The authentication factors can be something you know, such as a password, something you have, such as a token, and something you are, such as a fingerprint. Arguably, a key can form something you know and something you have at the same time, if a passphrase is set. Many people opt to not set a passphrase in order to make connecting easier. We would always recommend setting a passphrase on your SSH keys. In the case a key is compromised, though, something external like a phone or USB, can also provide an additional security step.

In this article I'm going to look at using Google Authenticator as a second security factor for securing SSH on your server.

Google Authenticator is an implementation of the Internet Engineering Task Force's RFC6238. As such, any other conforming software can be used, but Google have handily created a pluggable authentication module for Linux that can be used with SSH. Even more handily it's already pre-packaged in the Ubuntu repositories. The idea behind RFC6238 is that a key is created on the server side and is shared with the client. The client then creates a time-based one-time password based on that key, and at log-in time, that one-time password is checked for validity on the server. Each password is usually valid for 30 seconds.

So let's look at how we can implement this.First we need to install the Google Authenticator module on our Linux server…

sudo apt-get install libpam-google-authenticator

Once that is installed we then need to tell SSH how to use it. First we need to edit the Pluggable Athentication Module settings for SSH, which involves editing the file /etc/pam.d/sshd. In my example I'm using nano but you can substitute that for your preferred editor if you have one.

sudo nano /etc/pam.d/sshd

At the start of the file, add the following line, then save and exit.

auth required pam_google_authenticator.so

Once that is done we need to tell SSH to use that module. So we need to fire-up our favourite editor again, this time with the /etc/ssh/sshd_config file.

sudo nano /etc/ssh/sshd_config

This time we are looking for a line starting with ChallengeResponseAuthentication and need to set it to…

ChallengeResponseAuthentication yes

At this point we are ready to set up our account for the one-time password authentication. Next, you want to get the Google Authenticator application for your smartphone from the Android or Apple App Stores if you use one of those devices. Otherwise there are OTP applications available for the other smartphone platforms. Keep an eye out for it supporting RFC6238; it will work with the Google Authenticator module. Once this is installed on your phone we're ready to proceed to the next step.

On the server, as the user that needs to use two-factor authentication you need to run the following command:

google-authenticator

It will then ask you a few questions, such as 'would you like authentication to be time based?' I'd recommend saying yes.

Then it will display a QR code that you can scan with the authenticator app on your phone. If your screen can't display all of the code it will also give the secret key as a code you can type in.

Next it will ask if you want it to update your authenticator file. Again, say yes.

Now it will ask if you want to disallow multiple uses of the same authentication token. This will mean that, should your login attempt fail, you will need to wait for the 30 second expiry of that code in order to attempt to log-in again. This will slow down the logging in process, but will also slow down any attackers attempting to break in.

The next question will ask if you want to extend the window of valid codes in order to get in. The default should be fine unless your server's clock has a tendency to wander. If it does, I'd recommend installing NTP on there to make sure the times stay in sync.

Finally, you'll be asked if you want to enable rate limiting to 3 attempts every 30 seconds. Again, this is another useful security feature for blocking brute force attacks and I recommended that you say yes to it.

After this, you can try logging in over SSH with your new authenticator credentials. I’d recommend leaving the current session logged in and starting a new one, because if something goes wrong you are still connected and can disable or re-attempt setting up authenticator. If everything has worked then it first ask for your one-time password code and then your password. Once you have entered both in you should be successfully logged in.

Now you have to make sure you don’t lose your phone, or that you note down that secret key and lock it somewhere secure such as a safe or lockbox. As otherwise you’d not be able to log -n via SSH afterwards. You will still be able to gain local access to the server without the one-time password, so if your server has IPMI/iLO/etc you’ll still be able to gain access remotely that way in the case of an emergency should you lose access to your one-time passwords.

Related Articles

- Two-Factor authentication for SSH in CentOS 6 using Google…
- How To Secure A Server in 4 Simple Steps
- A VPN Primer

Save this article