Enabling Google 2FA

Overview

Two factor authentication is easy to configure and helps further secure your server. It requires a few extra packages -

```
sudo apt update && sude apt upgrade
sudo apt install libpam-google-authenticator
```

Along with these packages, we will need to make some changes to our `/etc/pam.d/sshd` and `/etc/ssh/sshd_config` files. See below for the complete steps.

Setup Google Authenticator

Run `google-authenticator` and respond to the prompts appropriately. Below is an example of the prompts output along with my responses - you can change or modify your responses to these prompts as you see fit. I did exclude some information for security reasons, but nothing more than the output between the prompts setting things up specific to my system.

```
Do you want authentication tokens to be time-based (y/n) y

...

Do you want me to update your "/home/host/.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, tokens are good for 30 seconds and 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. If you experience problems with poor
time synchronization, you can increase the window from its default
size of 1:30min to about 4min. Do you want to do so (y/n) n

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
```

The `...` within the code block above is a placeholder for information similar to the below -

```
Your new secret key is: XXXXXXXXXXXXXXXX
Your verification code is 123456
Your emergency scratch codes are:
  12345678
  23456789
  34567890
  45678901
  56789012
```

Yes, you *should* save those scratch codes. They act as static keys that can be used for 2FA in the event that you are unable to use the linked device for any reason. Along with this, a QR code will be output to your terminal, which can be easily scanned using the Google Authenticator application from any device. Alternatively, you could input your secret key into the application when creating a new token. This will give you an auto-regenerating token that has strong security features, such as the rate-limiting feature I enabled during the final prompt of the `google-authenticator` setup above. This will enable a timeout period between multiple failed login attempts, which makes it more difficult to brute-force.

> Pay attention to the output during the setup process, its important that this process is completed correctly. If it is not, you could face issues when attempting to SSH into your server. **If you are not careful, this could result in a lockout.**
> Always have a secondary login method, until you have verified that these settings work.

SSHD Configuration

> **SSHD** - Secure Shell Daemon
> **Daemon** - A long-running background process that answers requests for services.

In the final steps of the 2FA configuration process, we need to tell SSHD and PAM that we want to use 2FA during the login process. To start, SSHD needs to know that we wish to use custom authentication methods when a connection attempt is made. Add the following to `/etc/sshd/sshd_config`, and keep in mind that comments prefixed with an asterix(*) are custom comments that I've added to explain what we are doing when we change these settings. The rest of the comments found in these files are there by default, and will be included with any installation of SSHD or PAM.

```
# *Default value is 22, change this to whatever you wish and adjust firewall / iptables accordingly
# *This is not required to be changed for 2FA, but it is recommended for all public-facing SSHD configurations
# What ports, IPs and protocols we listen for
Port 1234

# *You should be using keys to authenticate / provision logins
PubkeyAuthentication yes

# *Since we use the above, you should not need to use passwords when logging in
# *If desired, this can still be enabled as an extra-layer
# *Password-only auth is easy to brute-force if not secured well
# Change to no to disable tunnelled clear text passwords
PasswordAuthentication no

# *Required for SSHD to allow the response to Verification code prompt on login attempt
# *This allows you to input and pass your 2FA code to the SSHD when logging in
# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication yes

# *Since we will be configuring PAM, make sure it is enabled
# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# be allowed through the ChallengeResponseAuthentication and
# PasswordAuthentication.  Depending on your PAM configuration,
# PAM authentication via ChallengeResponseAuthentication may bypass
# the setting of "PermitRootLogin yes
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# and ChallengeResponseAuthentication to 'no'.
UsePAM yes

# *Specify to SSHD what methods you want to use when a connection attempt is made
# *If this is not configured correctly, our PAM configuration could be ignored.
AuthenticationMethods publickey,keyboard-interactive
```

PAM Configuration

> **PAM** - Pluggable Authentication Modules

PAM allows us to add or configure our custom modules used during the authentication of various systems. For our needs, we will only be adding one line to the `/etc/pam.d/sshd file`. See the below for an example configuration, our change can be found within the first few lines prefixed by a custom comment that I've added.

```
# PAM configuration for the Secure Shell service

# Standard Un*x authentication.

# *Comment this line out to stop PAM from prompting for password on a connection attempt to SSHD
# *This should be configured according to your AllowPasswordAuthentication setting within /etc/ssh/sshd_config
#@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so

# *Add this line to require authetication via the google-authenticator module for PAM
auth    required        pam_google_authenticator.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account  required     pam_access.so

# Standard Un*x authorization.
@include common-account

# SELinux needs to be the first session rule.  This ensures that any
# lingering context has been cleared.  Without this it is possible that a
# module could execute code in the wrong domain.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so close

# Set the loginuid process attribute.
session    required     pam_loginuid.so

# Create a new session keyring.
session    optional     pam_keyinit.so force revoke

# Standard Un*x session setup and teardown.
@include common-session
```

```
# Print the message of the day upon successful login.
# This includes a dynamically generated part from /run/motd.dynamic
# and a static (admin-editable) part from /etc/motd.
session    optional     pam_motd.so  motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate

# Print the status of the user's mailbox upon successful login.
session    optional     pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session    required     pam_limits.so

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
session    required     pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were moved to
# /etc/default/locale, so read that as well.
session    required     pam_env.so user_readenv=1 envfile=/etc/default/locale

# SELinux needs to intervene at login time to ensure that the process starts
# in the proper default security context.  Only sessions which are intended
# to run in the user's context should be run after this.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so open

# Standard Un*x password updating.
@include common-password
```

The majority of the above file should be left alone, and **it is a sequential configuration** - this means that the order in which these settings are defined is important to how they are interpreted by PAM. If you wish to rearrange things you can, but be sure that you know what you are doing. The above configuration is verified working on several servers, in my experience at the time of this writing.

Be sure when you are changing these settings, you are running `sudo systemctl restart sshd.service ssh.service` to apply your changes, then try to login *from a new session*. There is no need to terminate your active session, or reload it. If you disconnect your session and you are unable to authenticate due to your changed settings, you could be in for a bad time.

Notes

>

> The `/etc/pam.d/common-auth` file does not need to be changed, but it is an interesting file to read if you have the time. I'll throw a snippet below since it is so short, but this file basically defines the authentication process used in `common-auth`, seen in the above `/etc/pam.d/sshd` configuration where we commented out the `@include common-auth` line.

Basically, this file defines how authentication is handled, and if you read below you can see that the `common-auth` module defaults to `pam_deny.so`, where the connection attempt is blocked by PAM. On a success, PAM simply sets `success=1`, which sequentially skips the `pam_deny.so` step and moves on to `pam_permit.so`, allowing the connection to take place.

```
# /etc/pam.d/common-auth - authentication settings common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of the authentication modules that define
# the central authentication scheme for use on the system
# (e.g., /etc/shadow, LDAP, Kerberos, etc.).  The default is to use the
# traditional Unix authentication mechanisms.
#
# As of pam 1.0.1-6, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
auth    [success=1 default=ignore]      pam_unix.so nullok_secure
# here's the fallback if no module succeeds
auth    requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
auth    required                        pam_permit.so
# and here are more per-package modules (the "Additional" block)
auth    optional                        pam_cap.so
# end of pam-auth-update config
```

Revision #2
Created 10 May 2019 14:38:27 by Shaun Reed
Updated 18 December 2021 16:37:39 by Shaun Reed