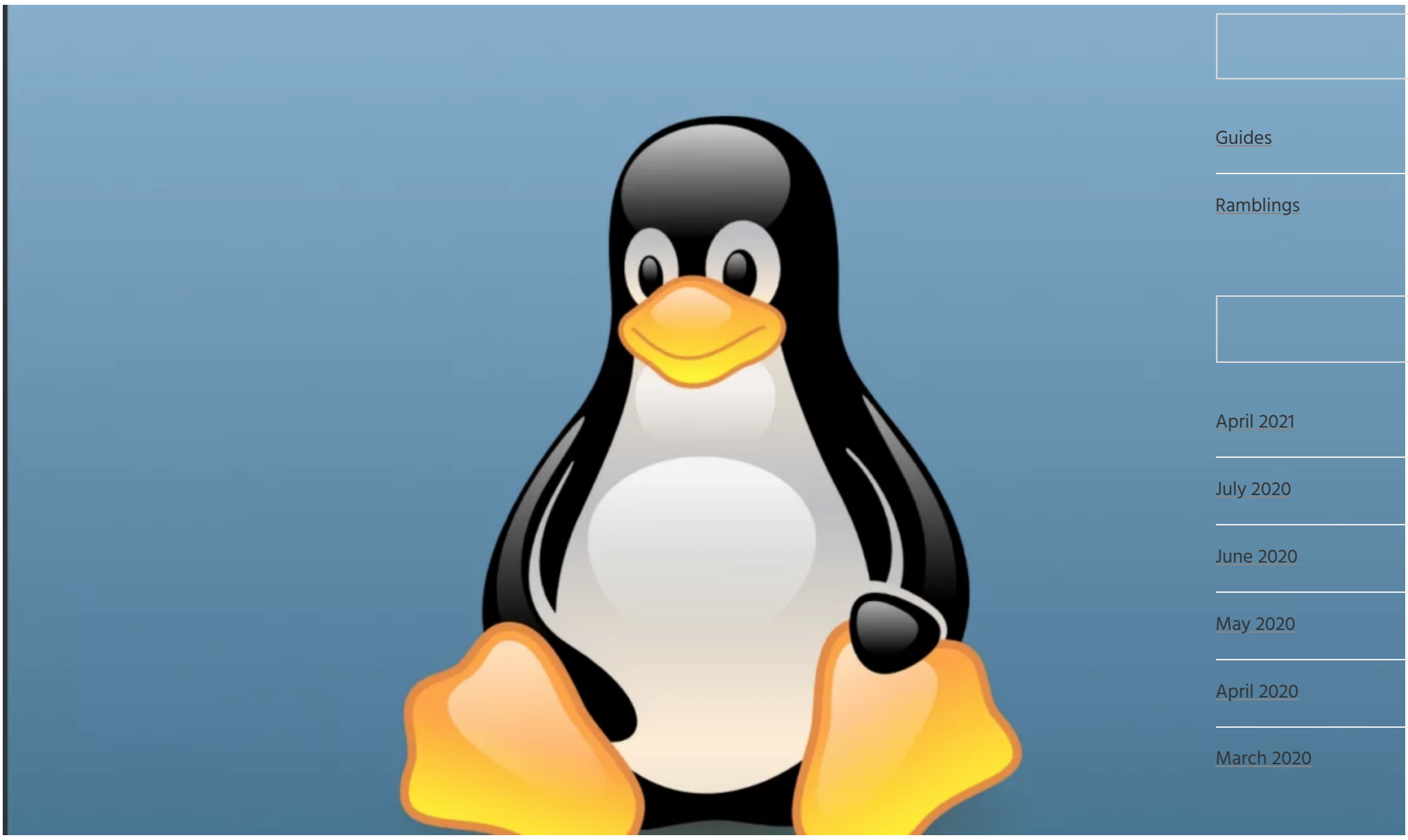
April 15, 2021

How to Automatically Sign Linux Kernel Modules After Kernel Update for Secure Boot



Every time I DuckDuckGoed anything remotely related to NVIDIA, Linux and secure boot, all I could ever find was "TURN IT OFF". But I was adamant, why would I turn off something that's supposed to make my workstation more secure? I am always looking to secure my stuff; I already use TPM, LUKS and Bitlocker, so why should I turn off secure boot every time I want to use a graphics card on Linux?

Being a Mechanical Engineer by degree and a Market Analyst by profession, everything I know about computers is a weird mixture of trial-and-error, common sense, and StackExchange. So, needless to say, proceed at your own risk.

Here's the directory structure we will be using for the script.

```
auto_kernel_signer
|- keys
    |- public_key.der
    |- private_key.priv
|- main.py
|- modules.conf
|- autosigner.log
|- configuration_file.config
```

First, we will make the parent directory (named auto_kernel_signer here), and the *keys* subdirectory. **Sign in to your root account using *sudo su* and *cd* to wherever you want to keep this script.**

```
~]# mkdir auto_kernel_signer && cd "$_"

~]# mkdir keys
```

Next, we will create new Machine Owner Key (MOK) and put the keypair in the keys subdirectory.

```
~]# cat << EOF > configuration_file.config
[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
O = Machine Owner
CN = Machine Owner signing key
emailAddress = root@localhost

[ myexts ]
basicConstraints=critical,CA:FALSE
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
EOF

~]# openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 \
 -batch -config configuration_file.config -outform DER \
 -out keys/public_key.der \
 -keyout keys/private_key.priv
```

Now that we have our key pairs, it's time to register the public key with our MOK Management.

```
~]# mokutil --import keys/public_key.der
```

You will be asked to enter and confirm a password for this MOK enrollment request. Restart the computer to trigger MOK Management, and import the key using the same password. Remember, mokutil does not work for everyone (although it does work most of the time). If it doesn't work for you, you will have to go into your UEFI firmware settings and import the public key directly. It should most likely be named MOK Management or some variations of that.

After we import the key to MOK database, the hard part (for you) is over! Copy the content from below and put it in main.py.

```
# Created by Meghadeep Roy Chowdhury 4/14/2021
# All rights reserved under GNU AGPLv3
# details: https://www.gnu.org/licenses/agpl-3.0.en.html

# Kernel sign script path
sign_script_path = '/usr/src/kernels/{uname_release}/scripts/sign-file'
# Common kernel path
path_common = '/lib/modules/'
# main.py directory path
main_path = '/home/meghadeep/PycharmProjects/auto_kernel_signer'
# shell script to list kernel versions available
shell_scr = 'rpm -q kernel | sort -V'

# Do not change below this line #
import os
import datetime
import subprocess
import sys


class MOKKeyError(Exception):
        """ Machine Owner's Key not found in the keys directory """
        pass


class SignError(Exception):
        """ Error while signing the kernel modules """
        pass


def prepend(modules, common):
        # Using format()
        common += '{0}'
        modules = [common.format(i) for i in modules]
        return modules


def sign_and_log(run_script, module):
        global main_path
        print('Signing ' + module)
        try:
                os.system(run_script)
                print('Signed ' + module)
                with open(main_path + '/autosigner.log', 'a+') as f:
                        f.write('Signed ' + module + '\n')
        except Exception as e:
                print('FAILURE: ' + module)
                print(e)
                with open(main_path + '/autosigner.log', 'a+') as f:
                        f.write(e + '\n')
                        f.write(datetime.datetime.now().strftime('%c') + '\n')


```

```python
def main():
        global sign_script_path

        # Get current kernel info
        kernel_current = os.uname().release

        # Get list of kernels
        kernels_present = subprocess.check_output(shell_scr, shell=True)
        kernels_present = kernels_present.decode().strip()
        # Get the most recently installed kernel
        if 'rpm' in shell_scr:
                kernel_updated = kernels_present.split('\n')[-1].split('kernel-')[-1]
        elif 'dpkg' in shell_scr:
                kernel_updated = kernels_present.split('\n')[-2].split('-image-')[-1]
        # Check if user is forcing signature
        try:
                override = sys.argv[1]
        except:
                override = False
        # Only need to proceed if there's been a kernel update
        if kernel_current != kernel_updated or override == 'force':
# if True:
                print('Updated kernel found: ' + kernel_updated)
                kernel_path = path_common + kernel_updated + '/'
                with open('modules.conf') as f:
                        kernel_modules = f.readlines()
                # Remove whitespace characters like `\n` at the end of each line
                if kernel_modules[-1] == '\n':
                        kernel_modules.pop()
                kernel_modules = [x.strip() for x in kernel_modules]
                modules_path = prepend(kernel_modules, kernel_path)
                # Check if keys exist
                keys = [f.name for f in os.scandir(main_path + '/keys') if f.is_file()
                if ('private_key.priv' in keys) and ('public_key.der' in keys):
                        print('Keys found in keys directory.')
                        public_key = main_path + '/keys/public_key.der'
                        private_key = main_path + '/keys/private_key.priv'
                        sign_script_path = sign_script_path.format(uname_release=kerne
                        for i in modules_path:
                                if i[-1] == '/':
                                        mod_list = os.listdir(i[:-1])
                                        for j in mod_list:
                                                run_script = sign_script_path + ' sha2
                                                sign_and_log(run_script, i + j)
                                else:
                                        run_script = sign_script_path + ' sha256 ' + p
                                        sign_and_log(run_script, i)
                else:
                        print('Keys NOT FOUND')
                        with open(main_path + '/autosigner.log', 'a+') as f:
                                f.write('Keys NOT FOUND. ' + datetime.datetime.now().s
                        raise MOKKeyError
        else:
                print('Kernel not updated, signing new kernels not required.')
                with open(main_path + '/autosigner.log', 'a+') as f:
                        f.write('Kernel not updated, signing new kernels not required.


if __name__ == '__main__':
        with open(main_path + '/autosigner.log', 'a+') as f:
                f.write('Service ran at ' + datetime.datetime.now().strftime('%c') + '

        main()
```

main.py hosted with ❤ by GitHub view raw

Pay close attention to the top four variables defined in the main.py file.

I am using Fedora, so if you're in the RHEL ecosystem, you probably won't have to change much.

The *main_path* variable is directory path where your main.py file resides.

For Ubuntu or other Debian based distros, the *sign_script_path* would be.

```
/usr/src/linux-headers-{uname_release}/scripts/sign-file
```

The *path_common* variable is defined for the kernel modules parent directory. You probably won't need to change that. Just make sure that's where your kernel modules reside for various kernel versions. So for example, if you want to sign the nvidia kernel and it's at:

```
/lib/modules/{kernel_version}/extra/nvidia/nvidia.ko
```

We will just keep the parent directory path of various kernel versions in our *path_common* variable as below.

```
/lib/modules/
```

The modules.conf file contains a list of modules you want to sign, but only relative paths to the updated kernel version. For example,

```
extra/nvidia/nvidia.ko
extra/nvidia/nvidia-drm.ko
extra/nvidia/nvidia-modeset.ko
extra/nvidia/nvidia-uvm.ko
```

Or, the directory as shown below, if you want to sign all the files in the specified directory

```
extra/nvidia/
```

So, in essence, your *path_common* variable will provide the base path for the modules you want to sign, the script will discover updated kernels in that directory, and modules.conf file will provide relative paths to the actual files that need to be signed within that updated kernel directory.

The *shell_scr* variable is the shell script that lists out installed kernel versions in ascending order. If you are in Ubuntu or other Debian based distros, you *shell_scr* would be:

```
dpkg --list | grep linux-image | grep ii | sort -V
```

After you make modules.conf, we will make a systemctl service that will run this script before shutdown.

```
~]# nano /lib/systemd/system/auto-kernel-signer.service
```

Then write the following:

```
[Unit]
Description=Auto Sign Kernel Modules

[Service]
User=root
Group=root
WorkingDirectory=/PATH_TO_SCRIPT_PARENT_DIR/auto_kernel_signe
r
Type=oneshot
RemainAfterExit=true
ExecStop=/usr/bin/python3
/PATH_TO_SCRIPT_PARENT_DIR/auto_kernel_signer/main.py

[Install]
WantedBy=multi-user.target
```

After saving the service, let's start it.

```
~]# systemctl daemon-reload

~]# systemctl enable auto-kernel-signer --now
```

Now that we have the service up and running, let's sign the kernel modules for the first time. If you're somehow doing this after a recent kernel upgrade and before the subsequent reboot, you won't have to manually sign the kernels.

Otherwise, *cd* into the *auto_kernel_signer* directory and run:

```
~]# python3 main.py force
```

And that's it! You're done. The service will run the script every time before your computer is turned off. The script checks if kernel has been updated. If there's no update, the script does nothing, if there's an update, the script signs the modules defined in modules.conf with the keys from keys subdirectory.

Enjoy secure booting Linux!

Share

Guides

Roy

An Indian expat learning to live 8000 miles away from home. Mechanical Engineer by degree, Market Analyst by profession.

2 Replies to "How to Automatically Sign Linux Kernel

Modules After Kernel Update for Secure Boot”

Muhammed | July 15, 2021 at 8:54 pm

I Have Trouble With Unsigned Linux Kernels After Enabling Secure Boot. I Am Using Pop OS 21.04. Will This Script Work?

Reply

Roy | August 29, 2021 at 11:02 pm

From What I Remember, PopOS! Does Not Officially Support Secure Boot, So, It Might Not. You Might Need To Sign Your Bootloader First To Get Secure Boot Working, But I May Be Wrong. If PopOS! Does Have A Signed Bootloader, In Which Case You Can Potentially Implement This Script, You Just Have To Edit The Required Directory Paths To Match Your Specific OS.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name * Email * Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Previous Reading
What’s wrong with Non-GMO products?