(/) Wiki

Signed kernel module support

From Gentoo Wiki

Since Linux kernel version 3.7 onwards, support has been added for signed kernel modules. When enabled, the Linux kernel will only load kernel modules that are digitally signed with the proper key. This allows further hardening of the system by disallowing unsigned kernel modules, or kernel modules signed with the wrong key, to be loaded. Malicious kernel modules are a common method for loading rootkits on a Linux system.

Contents

Enabling module signature verification

Enabling support is a matter of toggling a few settings in the Linux kernel configuration. Unless you want to use your own keypair, this is all that has to be done to enable kernel module support. It must also be noted that a change of file structure occurred in kernel version 4.3.3. So from this version and above the certificates and a sign files used to manually sign modules has moved into: `/usr/src/linux/certs/` Also the sign-file perl script is now directly executable in later kernel versions and does not require you to execute the `perl` command.

Adjust the installation and setup methods accordingly to the kernel version that will be used.

Configuring module signature verification

Module signature verification is a kernel feature, so has to be enabled through the Linux kernel configuration. You can find the necessary options under *Enable loadable module support*.

KERNEL **Enable module signature verification**

```
--- Enable loadable module support
[*]   Module signature verification
[*]     Require modules to be validly signed
[*]     Automatically sign all modules
      Which hash algorithm should modules be signed with? (Sign modules with SHA-512) --->
```

The option *Module signature verification* (*CONFIG_MODULE_SIG*) enables the module signature verification in the Linux kernel. It supports two approaches on signed module support: a rather permissive one and a strict one. By default, the permissive approach is used, which means that the Linux kernel module either has to have a valid signature, or no signature. With the strict approach, a valid signature must be present. In the above example, the strict approach is used by selecting *Require modules to be validly signed* (*CONFIG_MODULE_SIG_FORCE*). Another way of enabling this strict approach is to set the kernel boot option `enforcemodulesig=1` .

When building the Linux kernel, the kernel modules will not be signed automatically unless you select *Automatically sign all modules* (*CONFIG_MODULE_SIG_ALL*).

Finally, we need to select the hash algorithm to use with the cryptographic signature. In the above example, we use SHA-512.

Building the kernel with proper keys

When the Linux kernel is building with module signature verification support enabled, then you can use your own keys or have the Linux kernel build infrastructure create a set for you. If you want the Linux kernel build infrastructure to create it for you, just continue as you always do with a `make` and `make modules_install`. At the end of the build process, you will notice that `signing_key.priv` (`/certs/signing_key.pem` in kernel 4.3.3 or higher) and `signing_key.x509` will be available on the root of the Linux kernel sources.

If we want to use our own keys, you can use `openssl` to create a key pair (private key and public key). The following command, taken from `kernel/Makefile`, creates such a key pair.

Note: For kernels 4.3.3 and above x509.genkey is located in:

```
/usr/src/linux/certs/x509.genkey
```

FILE `x509.genkey` **Key generation configuration file**

```
[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
CN = Modules

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

```
user $ openssl req -new -nodes -utf8 -sha512 -days 36500 -batch -x509 -config x509.genkey -outform DER -out signing_key.x509
-keyout signing_key.priv
```

(`/certs/signing_key.pem` in kernel 4.3.3 or higher)
The resulting files need to be stored as `signing_key.x509` and `signing_key.priv` (`/certs/signing_key.pem` in kernel 4.3.3 or higher) in the root of the Linux kernel source tree.

The public key part will be build inside the Linux kernel. If you configured the kernel to sign modules, this signing will take place during the `make modules_install` part.

Validating module signature support

Reboot with the newly configured kernel. In the output of `dmesg` you should be able to confirm that the proper certificate is loaded:

```
root # dmesg | grep -i 'x.*509'
```

```
[    1.279874] Asymmetric key parser 'x509' registered
[    1.286431] Loading compiled-in X.509 certificates
[    1.292031] Loaded X.509 cert 'Modules: b03b5edb5700f9d5d785eb2d6f3e19d34a20205b'
```

If *CONFIG_KEYS_DEBUG_PROC_KEYS* is enabled, then root user can view certificate in `/proc/keys` file:

```
root # cat /proc/keys
```

```
011aee9c I------     1 perm 1f030000     0     0 asymmetri Modules: b03b5edb5700f9d5d785eb2d6f3e19d34a20205b: X509.RS
0b5726c7 I--Q---     1 perm 1f3f0000     0 65534 keyring   _uid_ses.0: 1
0b6539f7 I------     1 perm 1f0b0000     0     0 keyring   .system_keyring: 1
3b688504 I--Q---     2 perm 1f3f0000     0 65534 keyring   _uid.0: empty
```

`modinfo` should display same key:

```
user $ modinfo usbcore | grep '^sig'
```

```
signer:         Modules
sig_key:        B0:3B:5E:DB:57:00:F9:D5:D7:85:EB:2D:6F:3E:19:D3:4A:20:20:5B
sig_hashalgo:   sha512
```

The kernel modules have the digital signature appended at the end. A simple `hexdump` can confirm if a signature is present or not:

```
user $ hexdump -C vxlan.ko | tail
```

```
00008880  cf 0e e7 cb 10 9e 98 5f  4b 21 d4 03 ba 3d 7e e7  |......._K!...=~.|
00008890  68 db f9 e3 5f 62 3c c7  d6 6c 84 c7 d6 68 c1 73  |h..._b<..l...h.s|
000088a0  3d d7 5a 38 66 99 12 b8  84 c9 84 45 dd 68 6d 17  |=.Z8f......E.hm.|
000088b0  03 24 dc 9c 6f 6d 11 01  e9 74 82 ea b5 5b 46 07  |.$..om...t...[F.|
000088c0  fe dd 66 97 1a 33 58 3d  6e d0 ac 03 08 16 73 06  |..f..3X=n.....s.|
000088d0  9f 90 c4 eb b3 82 1d 9f  48 8c 5b 51 01 06 01 1e  |........H.[Q....|
000088e0  14 00 00 00 00 00 02 02  7e 4d 6f 64 75 6c 65 20  |........~Module |
000088f0  73 69 67 6e 61 74 75 72  65 20 61 70 70 65 6e 64  |signature append|
00008900  65 64 7e 0a                                       |ed~.|
00008904
```

The string `~Module signature appended~` at the end confirms that *a signature* is present. Of course, it does not confirm that the signature is valid or not.

To remove the signature, we can use the `strip` command:

```
root # strip --strip-debug vxlan.ko
root # hexdump -C vxlan.ko | tail
```

```
00097330  6c 5f 67 65 74 5f 73 74  61 74 73 36 34 00 72 63  |l_get_stats64.rc|
00097340  75 5f 62 61 72 72 69 65  72 00 5f 72 61 77 5f 73  |u_barrier._raw_s|
00097350  70 69 6e 5f 75 6e 6c 6f  63 6b 00 72 65 67 69 73  |pin_unlock.regis|
00097360  74 65 72 5f 70 65 72 6e  65 74 5f 64 65 76 69 63  |ter_pernet_devic|
00097370  65 00 6b 6d 61 6c 6c 6f  63 5f 63 61 63 68 65 73  |e.kmalloc_caches|
00097380  00 6e 65 74 64 65 76 5f  69 6e 66 6f 00 6e 65 69  |.netdev_info.nei|
00097390  67 68 5f 6c 6f 6f 6b 75  70 00 72 65 6c 65 61 73  |gh_lookup.releas|
000973a0  65 5f 73 6f 63 6b 00 72  65 67 69 73 74 65 72 5f  |e_sock.register_|
000973b0  6e 65 74 64 65 76 69 63  65 00                    |netdevice.|
000973ba
```

If we try to load this module now, we get a failure:

```
root # modprobe vxlan
```

```
modprobe: ERROR: could not insert 'vxlan': Required key not available
```

This confirms that modules without a signature are not loaded.

Administering kernel module signatures

Once the kernel boots and we have validated that the signed kernel module support works, it is important to correctly handle the keys themselves.

Protecting the private key

The private key, stored as `signing_key.priv` (`/certs/signing_key.pem` in kernel 4.3.3 or higher), needs to be moved to a secure location (unless you will be creating new keys for new kernels, in which case the file can be removed). Do not keep it at `/usr/src/linux` on production systems as malware can then easily use this key to sign the malicious kernel modules (such as rootkits) and compromise the system further.

The best way to do this is to move the key to a USB stick that you will unplug from your computer after it has been moved. That way if you update a driver such asd you graphics drivers which do not require a kernel re-compile you can insert the USB stick for that moment to sign your driver modules and unplug it after you have done so.

Other methods include technology such as a TPM yet these are not favored by many as the TPM module is burnt from manufacture with a key that the manufacturer has control over.

Manually signing modules

If you ever need to manually sign a kernel module, you can use the `scripts/sign-file` script available in the Linux kernel source tree. It requires four arguments:

1. The hash algorithm to use, such as `sha512`.
2. The private key location.
3. The certificate (which includes the public key) location.
4. The kernel module to sign.

In this case, the key pair does not need to be named `signing_file.priv` and such, nor do they need to be in the root of the Linux kernel source tree location.

```
root # perl /usr/src/linux/scripts/sign-file sha512 /usr/src/linux/kernel-signkey.priv /usr/src/linux/kernel-signkey.x509 vxlan.ko
```

For Kernel 4.3.3 or higher the perl script is no more. `"sign-file"` is an executable and its usage is as shown bellow, Note how the perl has been removed from the beginning of the command. The location of the certificates have also moved from the root source directory `/usr/src/linux` to the `/usr/src/linux/certs` directory and now uses `signing-key.pem` instead of `signing-key.priv`.

See command bellow:

```
root # /usr/src/linux/scripts/sign-file sha512 /usr/src/linux/certs/signing_key.pem /usr/src/linux/certs/signing_key.x509 vxlan.ko
```

Using the command for kernels prior to kernel version 4.3.3 on kernels after this version will produce error:

```
root # Unrecognized character \ ; marked by <-- HERE after <-- HERE near column 1 at /usr/src/linux/scripts/sign-file line 1.
```

Distributing the kernel and modules

If we create a kernel package through `make tarbz2-pkg`, the modules in it will be signed already so we do not need to manually sign them afterwards. The signing keys themselves are not distributed with it.

External resources

- Booting a self-signed Linux kernel (http://www.kroah.com/log/blog/2013/09/02/booting-a-self-signed-linux-kernel/) - Greg Kroah-Hartman describes how to boot a self-signed Linux kernel from EFI. As having signed kernel module support is only secure if the Linux kernel is trusted, this is an important (and related) feature to work with.

Retrieved from "http://wiki.gentoo.org/index.php?title=Signed_kernel_module_support&oldid=515496 (http://wiki.gentoo.org/index.php?title=Signed_kernel_module_support&oldid=515496)"

Categories (/wiki/Special:Categories): Server and Security (/wiki/Category:Server_and_Security) | Kernel (/wiki/Category:Kernel)

- This page was last modified on 13 July 2016, at 17:09.