Kernel module signing facility

Overview

The kernel module signing facility cryptographically signs modules during installation and then checks the signature upon loading the module. This allows increased kernel security by disallowing the loading of unsigned modules or modules signed with an invalid key. Module signing increases security by making it harder to load a malicious module into the kernel. The module signature checking is done by the kernel so that it is not necessary to have trusted userspace bits.

This facility uses X.509 ITU-T standard certificates to encode the public keys involved. The signatures are not themselves encoded in any industrial standard type. The facility currently only supports the RSA public key encryption standard (though it is pluggable and permits others to be used). The possible hash algorithms that can be used are SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 (the algorithm is selected by data in the signature).

Configuring module signing

The module signing facility is enabled by going to the :menuselection:`Enable Loadable Module Support` section of the kernel configuration and turning on:

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, **line 41**); ***backlink***
>
> Unknown interpreted text role "menuselection".

```
CONFIG_MODULE_SIG       "Module signature verification"
```

This has a number of options available:

1. :menuselection:`Require modules to be validly signed` (`CONFIG_MODULE_SIG_FORCE`)

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, **line 49**); ***backlink***
   >
   > Unknown interpreted text role "menuselection".

   This specifies how the kernel should deal with a module that has a signature for which the key is not known or a module that is unsigned.

   If this is off (ie. "permissive"), then modules for which the key is not available and modules that are unsigned are permitted, but the kernel will be marked as being tainted, and the concerned modules will be marked as tainted, shown with the character 'E'.

   If this is on (ie. "restrictive"), only modules that have a valid signature that can be verified by a public key in the kernel's possession will be loaded. All other modules will generate an error.

   Irrespective of the setting here, if the module has a signature block that cannot be parsed, it will be rejected out of hand.

2. :menuselection:`Automatically sign all modules` (`CONFIG_MODULE_SIG_ALL`)

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, **line 68**); ***backlink***
   >
   > Unknown interpreted text role "menuselection".

   If this is on then modules will be automatically signed during the modules_install phase of a build. If this is off, then the modules must be signed manually using:

   ```
   scripts/sign-file
   ```

3. :menuselection:`Which hash algorithm should modules be signed with?`

   > **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, **line 78**); ***backlink***
   >
   > Unknown interpreted text role "menuselection".

   This presents a choice of which hash algorithm the installation phase will sign the modules with:

`CONFIG_MODULE_SIG_SHA1`	:menuselection:`Sign modules with SHA-1` **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, **line 85**); ***backlink*** Unknown interpreted text role "menuselection".

`CONFIG_MODULE_SIG_SHA224`	:menuselection:`Sign modules with SHA-224` **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, line 86); *backlink* Unknown interpreted text role "menuselection".
`CONFIG_MODULE_SIG_SHA256`	:menuselection:`Sign modules with SHA-256` **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, line 87); *backlink* Unknown interpreted text role "menuselection".
`CONFIG_MODULE_SIG_SHA384`	:menuselection:`Sign modules with SHA-384` **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, line 88); *backlink* Unknown interpreted text role "menuselection".
`CONFIG_MODULE_SIG_SHA512`	:menuselection:`Sign modules with SHA-512` **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, line 89); *backlink* Unknown interpreted text role "menuselection".

The algorithm selected here will also be built into the kernel (rather than being a module) so that modules signed with that algorithm can have their signatures checked without causing a dependency loop.

4. :menuselection:`File name or PKCS#11 URI of module signing key` (`CONFIG_MODULE_SIG_KEY`)

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, line 96); *backlink*

   Unknown interpreted text role "menuselection".

   Setting this option to something other than its default of `certs/signing_key.pem` will disable the autogeneration of signing keys and allow the kernel modules to be signed with a key of your choosing. The string provided should identify a file containing both a private key and its corresponding X.509 certificate in PEM form, or â€" on systems where the OpenSSL ENGINE_pkcs11 is functional â€" a PKCS#11 URI as defined by RFC7512. In the latter case, the PKCS#11 URI should reference both a certificate and a private key.

   If the PEM file containing the private key is encrypted, or if the PKCS#11 token requires a PIN, this can be provided at build time by means of the `KBUILD_SIGN_PIN` variable.

5. :menuselection:`Additional X.509 keys for default system keyring` (`CONFIG_SYSTEM_TRUSTED_KEYS`)

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\(linux-master)(Documentation)(admin-guide)module-signing.rst`, line 113); *backlink*

   Unknown interpreted text role "menuselection".

   This option can be set to the filename of a PEM-encoded file containing additional certificates which will be included in the system keyring by default.

Note that enabling module signing adds a dependency on the OpenSSL devel packages to the kernel build processes for the tool that does the signing.

Generating signing keys

Cryptographic keypairs are required to generate and check signatures. A private key is used to generate a signature and the corresponding public key is used to check it. The private key is only needed during the build, after which it can be deleted or stored securely. The public key gets built into the kernel so that it can be used to check the signatures as the modules are loaded.

Under normal conditions, when `CONFIG_MODULE_SIG_KEY` is unchanged from its default, the kernel build will automatically generate a new keypair using openssl if one does not exist in the file:

```
certs/signing_key.pem
```

during the building of vmlinux (the public part of the key needs to be built into vmlinux) using parameters in the:

```
certs/x509.genkey
```

file (which is also generated if it does not already exist).

It is strongly recommended that you provide your own x509.genkey file.

Most notably, in the x509.genkey file, the req_distinguished_name section should be altered from the default:

```
[ req_distinguished_name ]
#O = Unspecified company
CN = Build time autogenerated kernel key
#emailAddress = unspecified.user@unspecified.company
```

The generated RSA key size can also be set with:

```
[ req ]
default_bits = 4096
```

It is also possible to manually generate the key private/public files using the x509.genkey key generation configuration file in the root node of the Linux kernel sources tree and the openssl command. The following is an example to generate the public/private key files:

```
openssl req -new -nodes -utf8 -sha256 -days 36500 -batch -x509 \
   -config x509.genkey -outform PEM -out kernel_key.pem \
   -keyout kernel_key.pem
```

The full pathname for the resulting kernel_key.pem file can then be specified in the `CONFIG_MODULE_SIG_KEY` option, and the certificate and key therein will be used instead of an autogenerated keypair.

Public keys in the kernel

The kernel contains a ring of public keys that can be viewed by root. They're in a keyring called ".builtin_trusted_keys" that can be seen by:

```
[root@deneb ~]# cat /proc/keys
...
223c7853 I------     1 perm 1f030000     0     0 keyring   .builtin_trusted_keys: 1
302d2d52 I------     1 perm 1f010000     0     0 asymmetri Fedora kernel signing key: d69a84e6bce3d216b979e9505b3e3ef9a711807
...
```

Beyond the public key generated specifically for module signing, additional trusted certificates can be provided in a PEM-encoded file referenced by the `CONFIG_SYSTEM_TRUSTED_KEYS` configuration option.

Further, the architecture code may take public keys from a hardware store and add those in also (e.g. from the UEFI key database).

Finally, it is possible to add additional public keys by doing:

```
keyctl padd asymmetric "" [.builtin_trusted_keys-ID] <[key-file]
```

e.g.:

```
keyctl padd asymmetric "" 0x223c7853 <my_public_key.x509
```

Note, however, that the kernel will only permit keys to be added to `.builtin_trusted_keys` **if** the new key's X.509 wrapper is validly signed by a key that is already resident in the `.builtin_trusted_keys` at the time the key was added.

Manually signing modules

To manually sign a module, use the scripts/sign-file tool available in the Linux kernel source tree. The script requires 4 arguments:

1. The hash algorithm (e.g., sha256)
2. The private key filename or PKCS#11 URI
3. The public key filename
4. The kernel module to be signed

The following is an example to sign a kernel module:

```
scripts/sign-file sha512 kernel-signkey.priv \
        kernel-signkey.x509 module.ko
```

The hash algorithm used does not have to match the one configured, but if it doesn't, you should make sure that hash algorithm is either built into the kernel or can be loaded without requiring itself.

If the private key requires a passphrase or PIN, it can be provided in the $KBUILD_SIGN_PIN environment variable.

Signed modules and stripping

A signed module has a digital signature simply appended at the end. The string `~Module signature appended~.` at the end of the module's file confirms that a signature is present but it does not confirm that the signature is valid!

Signed modules are BRITTLE as the signature is outside of the defined ELF container. Thus they MAY NOT be stripped once the signature is computed and attached. Note the entire module is the signed payload, including any and all debug information present at the time of signing.

Loading signed modules

Modules are loaded with insmod, modprobe, `init_module()` or `finit_module()`, exactly as for unsigned modules as no processing is done in userspace. The signature checking is all done within the kernel.

Non-valid signatures and unsigned modules

If `CONFIG_MODULE_SIG_FORCE` is enabled or module.sig_enforce=1 is supplied on the kernel command line, the kernel will only load validly signed modules for which it has a public key. Otherwise, it will also load modules that are unsigned. Any module for which the kernel has a key, but which proves to have a signature mismatch will not be permitted to load.

Any module that has an unparseable signature will be rejected.

Administering/protecting the private key

Since the private key is used to sign modules, viruses and malware could use the private key to sign modules and compromise the operating system. The private key must be either destroyed or moved to a secure location and not kept in the root node of the kernel source tree.

If you use the same private key to sign modules for multiple kernel configurations, you must ensure that the module version information is sufficient to prevent loading a module into a different kernel. Either set `CONFIG_MODVERSIONS=y` or ensure that each configuration has a different kernel release string by changing `EXTRAVERSION` or `CONFIG_LOCALVERSION`.