module-signing

Signature des modules kernel

Présentation

La facilité de signature de module kernel signe cryptographiquement les modules durant l'installation et vérifie la signature au chargement des modules. Cela augmente la sécurité de kernel en interdisant le chargement des modules non-signés ou les modules signés avec une clé invalide. La signature de module augmente la sécurité en rendant le chargement de code malicieux plus dur dans le kernel. La vérification de la signature est fait par le kernel donc il n'est pas nécessaire d'avoir de clé de confiance dans l'espace utilisateur.

Cette facilité utilise les certificats X.509 pour encoder les clé publiques. Les signature ne sont pas encodées par un type standard. La facilité supporte actuellement seulement RSA. Les algorithmes de hashage qui peuvent être utilisée sont SHA-1, SHA-224, SHA-256, SHA-384, et SHA-512.

Configuration

La facilité de signature de module est activée par la section "Enable Loadable Module Support" de la configuration du kernel et en activant CONFIG_MODULE_SIG. Il y a quelques options disponibles :

Require modules to be validly signed CONFIG_MODULE_SIG_FORCE - spécifie le comportement du kernel lorsque la signature d'un module ne peut pas être vérifiée. off, autorise les module non-signés ou non valides, restrictive impose une signature valide.

Automatically sign all modules CONFIG_MODULE_SIG_ALL - Signe les modules automatiquement durant la phase modules_install. À off, les modules doivent être signés manuellement avec scripts/sign-file

Which hash algorithm should modules be signed with ? Présente un choix d'algorithmes de hashage pour signer les modules

File name or PKCS#11 URI of module signing key CONFIG_MODULE_SIG_KEY - Définir cette option à une valeur différente que le défaut "certs/signing_key.pem" désactive l'autogénération des clés de signatures. Cette chaîne devrait identifier un fichier contenant une clé privée et son certificat correspondant au format PEM, ou - sur les système où OpenSSL ENGINE pkcs11 est fonctionnel

Additional X.509 keys for default system keyring CONFIG_SYSTEM_TRUSTED_KEYS - Définis un fichier PEM contenant des certificats additionnels qui seront inclus dans le système par défaut.

Générer les clés

Une paire de clé cryptographique est requise pour générer et vérifier les signatures. Une clé privée est utilisée pour générer une signature et la clé publique correspondante est utilisée pour la vérifier. La clé privée est seulement nécessaire durant le build, et peut être supprimée ensuite. La clé publique est embarquée dans le kernel.

Dans des conditions normales, CONFIG_MODULE_SIG_KEY est inchangé de son défaut, le build génère automatiquement une nouvelle clé en utilisant openssl s'il n'existe pas de fichier certs/signing_key.pem. certs/x509.genkey est également généré et contient les paramètres openssl.

Par défaut la section req_distinguished_name contient :

```
[ req_distinguished_name ]
```

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

La taille de clé est également définie avec :

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

Il est également possible de générer manuellement les fichiers avec le fichier x509.genkey :
openssl req -new -nodes -utf8 -sha256 -days 36500 -batch -x509 -config x509.genkey -outform PEM -out kernel_key.pem -keyout kernel_key.pem

Clés publiques dans le kernel

Le kernel contient un jeu de clé publiques qui peuvent être vues par root (cat /proc/keys).

Au delà la clé publique générée spécifiquement pour la signature de module, des certificats additionnels peuvent être fournis au format PEM référencés par l'option kernel CONFIG_SYSTEM_TRUSTED_KEYS.

De plus, le code peut prendre des clés publiques depuis un stockage hardware et les ajouter (ex : depuis une base de clé UEFI)

Finalement, il est possible d'ajouter des clé publiques en faisant :

Noter cependant que le kernel ne permet que d'ajouter des clés dans .system_keyring s'ils sont signés par une clé déjà présente dans le .system_keyring au moment où elle est ajoutée.

Signer des modules manuellement

Pour signer manuellement un module, utiliser le script sign-file. Ce script nécessite 4 arguments : l'algorithme de hashage, le nom de la clé privée ou une URI PKCS#11, la clé publique et le module à signer.

exemple :
scripts/sign-file sha512 kernel-signkey.priv kernel-signkey.x509 module.ko
L'algorithme de hashage utilisé ne doit pas correspondre à celui configuré, mais l'algorithme utilisé doit être compilé dans le kernel ou un module chargeable. Si la clé privée nécessite une passphrase ou PIN, il peut être fournis dans la variable d'environnement $KBUILD_SIGN_PIN

Modules signés et stripping

Un module signé a une signature numérique simplement ajouté à la fin. La chaîne '~Module signature appended~.' à la fin diu fichier du module confirme qu'une signature est présente mais ne confirme pas que la signature est valide.

Les modules signés sont embarqué comme signature hors du conteneur ELF. Donc, ils ne peuvent plus être stripped une fois la signature calculée. Noter que tout le module est le payload signé, incluant les informations de debug présents au moment de la signature.

Charger des modules signés

Les modules sont chargés avec insmod, modprobe, init_module() ou finit_module(), exactement comme pour les modules non-signés et aucun traitement n'est fait dans l'espace utilisateur. La vérification de la signature est entièrement faite dans le kernel

Signatures non-valides et modules non-signé

Si CONFIG_MODULE_SIG_FORCE est activé ou enforcemodulesig=1 est fournis au kernel, le kernel va seulement charger les modules signés et valides pour lequel il possède une clé publique. Un module pour lequel le kernel a une clé, mais qui prouve qu'une signature ne correspond pas ne sera pas chargé. Tout module dont la signature n'est pas lisible sera rejeté.

Administrer/protéger la clé privée

Vu que la clé privée est utilisée pour signer les modules, les virus et malware peuvent l'utiliser pour signer des modules et compromettre l'OS. La clé privée doit être soit détruite, soit placée dans une emplacement sûre et non possédé dans le nœud root du kernel.