Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'evmctl.1' command

$ man evmctl.1

EVMCTL(1) EVMCTL(1)

NAME

evmctl - IMA/EVM signing utility

SYNOPSIS

evmctl [options] <command> [OPTIONS]

DESCRIPTION

The evmctl utility can be used for producing and verifying digital signatures, which are used by Linux kernel integrity subsystem (IMA/EVM). It can be also used to import keys into the kernel keyring.

COMMANDS

--version

help <command>

import [--rsa] pubkey keyring

sign [-r] [--imahash | --imasig] [--portable] [--key key] [--pass password] file

verify file

ima_boot_aggregate [--pcrs hash-algorithm,file] [TPM 1.2 BIOS event log]

ima_sign [--sigfile] [--key key] [--pass password] file

ima_verify file

ima_hash file

ima_measurement [--ignore-violations] [--verify-sig [--key "key1, key2, ..."]] [--pcrs [hash-algorithm,]file [--pcrs hash-algorithm,file] ...] file

ima_fix [-t fdsxm] path

sign_hash [--key key] [--pass password]

```
  hmac [--imahash | --imasig ] file

OPTIONS
  -a, --hashalgo     sha1, sha224, sha256, sha384, sha512
  -s, --imasig       make IMA signature
  -d, --imahash      make IMA hash
  -f, --sigfile      store IMA signature in .sig file instead of xattr
      --xattr-user   store xattrs in user namespace (for testing purposes)
      --rsa          use RSA key type and signing scheme v1
  -k, --key          path to signing key (default: /etc/keys/{privkey,pubkey}_evm.pem)
                     or a pkcs11 URI
      --keyid n      overwrite signature keyid with a 32-bit value in hex (for signing)
      --keyid-from-cert file
                     read keyid value from SKID of a x509 cert file
  -o, --portable     generate portable EVM signatures
  -p, --pass         password for encrypted signing key
  -r, --recursive    recurse into directories (sign)
  -t, --type         file types to fix 'fxm' (f: file)
                     x - skip fixing if both ima and evm xattrs exist (use with caution)
                     m - stay on the same filesystem (like 'find -xdev')
  -n                 print result to stdout instead of setting xattr
  -u, --uuid         use custom FS UUID for EVM (unspecified: from FS, empty: do not use)
      --smack        use extra SMACK xattrs for EVM
      --m32          force EVM hmac/signature for 32 bit target system
      --m64          force EVM hmac/signature for 64 bit target system
      --engine e     preload OpenSSL engine e (such as: gost)
      --pcrs         file containing TPM pcrs, one per hash-algorithm/bank
      --ignore-violations ignore ToMToU measurement violations
      --verify-sig   verify the file signature based on the file hash, both
                     stored in the template data.
  -v                 increase verbosity level
  -h, --help         display this help and exit

 Environment variables:

 EVMCTL_KEY_PASSWORD : Private key password to use; do not use --pass
```

option

INTRODUCTION

Linux kernel integrity subsystem is comprised of a number of different components including the Integrity Measurement Architecture (IMA), Extended Verification Module (EVM), IMA-appraisal extension, digital signature verification extension and audit measurement log support.

The evmctl utility is used for producing and verifying digital signatures, which are used by the Linux kernel integrity subsystem. It is also used for importing keys into the kernel keyring.

Linux integrity subsystem allows to use IMA and EVM signatures. EVM signature protects file metadata, such as file attributes and extended attributes. IMA signature protects file content.

For more detailed information about integrity subsystem it is recommended to follow resources in RESOURCES section.

EVM HMAC AND SIGNATURE METADATA

EVM protects file metadata by including following attributes into HMAC and signature calculation: inode number, inode generation, UID, GID, file mode, security.selinux, security.SMACK64, security.ima, security.capability.

EVM HMAC and signature in may also include additional file and file system attributes. Currently supported additional attributes are filesystem UUID and extra SMACK extended attributes.

Kernel configuration option CONFIG_EVM_ATTR_FSUUID controls whether to include filesystem UUID into HMAC and enabled by default. Therefore evmctl also includes fsuuid by default. Providing --uuid option without parameter allows to disable usage of fs uuid. Providing --uuid=UUID option with parameter allows to use custom UUID. Providing the --portable option will disable usage of the fs uuid and also the inode number and generation.

Kernel configuration option CONFIG_EVM_EXTRA_SMACK_XATTRS controls whether to include additional SMACK extended attributes into HMAC. They are following: security.SMACK64EXEC, security.SMACK64TRANSMUTE and security.SMACK64MMAP. evmctl --smack options enables that.

KEY AND SIGNATURE FORMATS

Linux integrity subsystem supports two type of signature and respectively two key formats.

First key format (v1) is pure RSA key encoded in PEM a format and uses own signature format. It is now non-default format and requires to provide evmctl --rsa option for signing and importing the key.

Second key format uses X509 DER encoded public key certificates and uses asymmetric key support in the kernel (since kernel 3.9). CONFIG_INTEGRITY_ASYMMETRIC_KEYS must be enabled (default).

For v2 signatures x509 certificate (containing the public key) could be appended to the private key (they both are in PEM format) to automatically extract keyid from its Subject Key Identifier (SKID).

INTEGRITY KEYRINGS

Integrity subsystem uses dedicated IMA/EVM keyrings to search for signature verification keys - _ima and _evm respectively.

Since 3.13 IMA allows to declare IMA keyring as trusted. It allows only to load keys, signed by a key from the system keyring (.system). It means self-signed keys are not allowed. This is a default behavior unless CONFIG_IMA_TRUSTED_KEYRING is undefined. IMA trusted keyring is has different name .ima. Trusted keyring requires X509 public key certificates. Old version RSA public keys are not compatible with trusted keyring.

GENERATE EVM ENCRYPTED KEYS

EVM encrypted key is used for EVM HMAC calculation:

```
# create and save the key kernel master key (user type)
# LMK is used to encrypt encrypted keys
keyctl add user kmk "`dd if=/dev/urandom bs=1 count=32 2>/dev/null`" @u
keyctl pipe `keyctl search @u user kmk` > /etc/keys/kmk
# create the EVM encrypted key
keyctl add encrypted evm-key "new user:kmk 64" @u
keyctl pipe `keyctl search @u encrypted evm-key` >/etc/keys/evm-key
```

GENERATE EVM TRUSTED KEYS (TPM BASED)

Trusted EVM keys are keys which a generate with the help of TPM. They are not related to integrity trusted keys.

```
# create and save the key kernel master key (user type)

keyctl add trusted kmk "new 32" @u

keyctl pipe `keyctl search @u trusted kmk` >kmk

# create the EVM trusted key

keyctl add encrypted evm-key "new trusted:kmk 32" @u

keyctl pipe `keyctl search @u encrypted evm-key` >evm-key
```

GENERATE SIGNING AND VERIFICATION KEYS

Generate private key in plain text format:

```
openssl genrsa -out privkey_evm.pem 1024
```

Generate encrypted private key:

```
openssl genrsa -des3 -out privkey_evm.pem 1024
```

Make encrypted private key from unencrypted:

```
openssl rsa -in /etc/keys/privkey_evm.pem -out privkey_evm_enc.pem -des3
```

Generate self-signed X509 public key certificate and private key for

using kernel asymmetric keys support:

```
openssl req -new -nodes -utf8 -sha1 -days 36500 -batch \

        -x509 -config x509_evm.genkey \

        -outform DER -out x509_evm.der -keyout privkey_evm.pem
```

Configuration file x509_evm.genkey:

```
# Begining of the file

[ req ]

default_bits = 1024

distinguished_name = req_distinguished_name

prompt = no

string_mask = utf8only

x509_extensions = myexts

[ req_distinguished_name ]

O = Magrathea

CN = Glacier signing key

emailAddress = slartibartfast@magrathea.h2g2

[ myexts ]

basicConstraints=critical,CA:FALSE
```

```
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
# EOF
```

Generate public key for using RSA key format:

```
openssl rsa -pubout -in privkey_evm.pem -out pubkey_evm.pem
```

Copy keys to /etc/keys:

```
  cp pubkey_evm.pem /etc/keys
  scp pubkey_evm.pem target:/etc/keys
or
  cp x509_evm.pem /etc/keys
  scp x509_evm.pem target:/etc/keys
```

GENERATE TRUSTED KEYS

Generation of trusted keys is a bit more complicated process and involves following steps:

? Creation of local IMA certification authority (CA). It consist of private and public key certificate which are used to sign and verify other keys.

? Build Linux kernel with embedded local IMA CA X509 certificate. It is used to verify other keys added to the .ima trusted keyring

? Generate IMA private signing key and verification public key certificate, which is signed using local IMA CA private key.

Configuration file ima-local-ca.genkey:

```
# Begining of the file
[ req ]
default_bits = 2048
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = v3_ca
[ req_distinguished_name ]
O = IMA-CA
CN = IMA/EVM certificate signing key
```

```
emailAddress = ca@ima-ca

[ v3_ca ]

basicConstraints=CA:TRUE

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid:always,issuer

# keyUsage = cRLSign, keyCertSign

# EOF
```

Generate private key and X509 public key certificate:

```
openssl req -new -x509 -utf8 -sha1 -days 3650 -batch -config $GENKEY \

            -outform DER -out ima-local-ca.x509 -keyout ima-local-ca.priv
```

Produce X509 in DER format for using while building the kernel:

```
openssl x509 -inform DER -in ima-local-ca.x509 -out ima-local-ca.pem
```

Configuration file ima.genkey:

```
# Begining of the file

[ req ]

default_bits = 1024

distinguished_name = req_distinguished_name

prompt = no

string_mask = utf8only

x509_extensions = v3_usr

[ req_distinguished_name ]

O = `hostname`

CN = `whoami` signing key

emailAddress = `whoami`@`hostname`

[ v3_usr ]

basicConstraints=critical,CA:FALSE

#basicConstraints=CA:FALSE

keyUsage=digitalSignature

#keyUsage = nonRepudiation, digitalSignature, keyEncipherment

subjectKeyIdentifier=hash

authorityKeyIdentifier=keyid

#authorityKeyIdentifier=keyid,issuer

# EOF
```

Generate private key and X509 public key certificate signing request:

```
openssl req -new -nodes -utf8 -sha1 -days 365 -batch -config $GENKEY \
            -out csr_ima.pem -keyout privkey_ima.pem
```

Sign X509 public key certificate signing request with local IMA CA

private key:

```
openssl x509 -req -in csr_ima.pem -days 365 -extfile $GENKEY -extensions v3_usr \
             -CA ima-local-ca.pem -CAkey ima-local-ca.priv -CAcreateserial \
             -outform DER -out x509_ima.der
```

SIGN FILE DATA AND METADATA

Default key locations:

```
Private RSA key: /etc/keys/privkey_evm.pem
Public RSA key: /etc/keys/pubkey_evm.pem
X509 certificate: /etc/keys/x509_evm.der
```

Options to remember: -k, -r, --rsa, --uuid, --smack.

Sign file with EVM signature and calculate hash value for IMA:

```
evmctl sign --imahash test.txt
```

Sign file with both IMA and EVM signatures:

```
evmctl sign --imasig test.txt:
```

Sign file with IMA signature:

```
evmctl ima_sign test.txt
```

Sign recursively whole filesystem:

```
evmctl -r sign --imahash /
```

Fix recursively whole filesystem:

```
evmctl -r ima_fix /
```

Sign filesystem selectively using find command:

```
find / \( -fstype rootfs -o -fstype ext4 \) -exec evmctl sign --imahash '{}' \;
```

Fix filesystem selectively using find command:

```
find / \( -fstype rootfs -o -fstype ext4 \) -exec sh -c "< '{}'" \;
```

INITIALIZE IMA/EVM AT EARLY BOOT

IMA/EVM initialization should be normally done from initial RAM file system before mounting root filesystem.

Here is Ubuntu initramfs example script

(/etc/initramfs-tools/scripts/local-top/ima.sh)

```
# mount securityfs if not mounted
SECFS=/sys/kernel/security
grep -q  $SECFS /proc/mounts || mount -n -t securityfs securityfs $SECFS
# search for IMA trusted keyring, then for untrusted
ima_id="`awk '/\.ima/ { printf "%d", "0x"$1; }' /proc/keys`"
if [ -z "$ima_id" ]; then
    ima_id=`keyctl search @u keyring _ima 2>/dev/null`
    if [ -z "$ima_id" ]; then
        ima_id=`keyctl newring _ima @u`
    fi
fi
# import IMA X509 certificate
evmctl import /etc/keys/x509_ima.der $ima_id
# search for EVM keyring
evm_id=`keyctl search @u keyring _evm 2>/dev/null`
if [ -z "$evm_id" ]; then
    evm_id=`keyctl newring _evm @u`
fi
# import EVM X509 certificate
evmctl import /etc/keys/x509_evm.der $evm_id
# a) import EVM encrypted key
cat /etc/keys/kmk | keyctl padd user kmk @u
keyctl add encrypted evm-key "load `cat /etc/keys/evm-key`" @u
# OR
# b) import EVM trusted key
keyctl add trusted kmk "load `cat /etc/keys/kmk`" @u
keyctl add encrypted evm-key "load `cat /etc/keys/evm-key`" @u
# enable EVM
echo "1" > /sys/kernel/security/evm
```

Optionally it is possible also to forbid adding, removing of new public keys and certificates into keyrings and revoking keys using keyctl setperm command:

```
# protect EVM keyring
```

```
keyctl setperm $evm_id 0x0b0b0000

# protect IMA keyring

keyctl setperm $ima_id 0x0b0b0000

# protecting IMA key from revoking (against DoS)

ima_key=`evmctl import /etc/keys/x509_ima.der $ima_id`

keyctl setperm $ima_key 0x0b0b0000
```

When using plain RSA public keys in PEM format, use evmctl import --rsa

for importing keys:

```
evmctl import --rsa /etc/keys/pubkey_evm.pem $evm_id
```

Latest version of keyctl allows to import X509 public key certificates:

```
cat /etc/keys/x509_ima.der | keyctl padd asymmetric '' $ima_id
```

FILES

Examples of scripts to generate X509 public key certificates:

```
/usr/share/doc/ima-evm-utils/ima-genkey-self.sh

/usr/share/doc/ima-evm-utils/ima-genkey.sh

/usr/share/doc/ima-evm-utils/ima-gen-local-ca.sh
```

AUTHOR

Written by Dmitry Kasatkin, <dmitry.kasatkin at gmail.com> and others.

RESOURCES

http://sourceforge.net/p/linux-ima/wiki/Home

http://sourceforge.net/p/linux-ima/ima-evm-utils

COPYING

Copyright (C) 2012 - 2014 Linux Integrity Project. Free use of this

software is granted under the terms of the GNU Public License (GPL).

12/13/2021 EVMCTL(1)