MyWebUniversity
python
Red Hat Enterprise Linux Manual Pages
FPDF Library PDF generator
FPDF
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 'containers-policy.json.5' command

$ man containers-policy.json.5

CONTAINERS-POLICY.JSON(5)   Page   CONTAINERS-POLICY.JSON(5)

Miloslav Trma? September 2016

NAME

containers-policy.json - syntax for the signature verification policy

file

DESCRIPTION

Signature verification policy files are used to specify policy, e.g.

trusted keys, applicable when deciding whether to accept an image, or

individual signatures of that image, as valid.

By default, the policy is read from $HOME/.config/containers/pol?

icy.json, if it exists, otherwise from /etc/containers/policy.json;

applications performing verification may allow using a different policy

instead.

FORMAT

The signature verification policy file, usually called policy.json,

uses a JSON format. Unlike some other JSON files, its parsing is

fairly strict: unrecognized, duplicated or otherwise invalid fields

cause the entire file, and usually the entire operation, to be re?

jected.

The purpose of the policy file is to define a set of policy require?

ments for a container image, usually depending on its location (where

it is being pulled from) or otherwise defined identity.

Policy requirements can be defined for:

? An individual scope in a transport. The transport values are
the same as the transport prefixes when pushing/pulling images
(e.g. docker:, atomic:), and scope values are defined by each
transport; see below for more details.

Usually, a scope can be defined to match a single image, and various
prefixes of
such a most specific scope define namespaces of matching images.

? A default policy for a single transport, expressed using an
empty string as a scope

? A global default policy.

If multiple policy requirements match a given image, only the require?
ments from the most specific match apply, the more general policy re?
quirements definitions are ignored.

This is expressed in JSON using the top-level syntax

```
{
    "default": [/* policy requirements: global default */]
    "transports": {
        transport_name: {
            "": [/* policy requirements: default for transport $transport_name */],
            scope_1: [/* policy requirements: default for $scope_1 in $transport_name */],
            scope_2: [/*?*/]
            /*?*/
        },
        transport_name_2: {/*?*/}
        /*?*/
    }
}
```

The global default set of policy requirements is mandatory; all of the
other fields (transports itself, any specific transport, the transport-
specific default, etc.) are optional.

Supported transports and their scopes

atomic:

The atomic: transport refers to images in an Atomic Registry.

Supported scopes use the form hostname[:port][/namespace[/imagestream
[:tag]]], i.e. either specifying a complete name of a tagged image, or
prefix denoting a host/namespace/image stream or a wildcarded expres?
sion for matching all subdomains. For wildcarded subdomain matching,
.example.com is a valid case, but example.*.com is not.
Note: The hostname and port refer to the container registry host and
port (the one used e.g. for docker pull), not to the OpenShift API host
and port.

dir:

The dir: transport refers to images stored in local directories.
Supported scopes are paths of directories (either containing a single
image or subdirectories possibly containing images).
Note: The paths must be absolute and contain no symlinks. Paths violat?
ing these requirements may be silently ignored.
The top-level scope "/" is forbidden; use the transport default scope
"", for consistency with other transports.

docker:

The docker: transport refers to images in a registry implementing the
"Docker Registry HTTP API V2".
Scopes matching individual images are named Docker references in the
fully expanded form, either using a tag or digest. For example,
docker.io/library/busybox:latest (not busybox:latest).
More general scopes are prefixes of individual-image scopes, and spec?
ify a repository (by omitting the tag or digest), a repository name?
space, or a registry host (by only specifying the host name) or a wild?
carded expression for matching all subdomains. For wildcarded subdomain
matching, *.example.com is a valid case, but example*.*.com is not.

oci:

The oci: transport refers to images in directories compliant with "Open
Container Image Layout Specification".
Supported scopes use the form directory:tag, and directory referring to
a directory containing one or more tags, or any of the parent directo?
ries.

Note: See dir: above for semantics and restrictions on the directory paths, they apply to oci: equivalently.

tarball:

The tarball: transport refers to tarred up container root filesystems.

Scopes are ignored.

Policy Requirements

Using the mechanisms above, a set of policy requirements is looked up. The policy requirements are represented as a JSON array of individual requirement objects. For an image to be accepted, all of the require? ments must be satisfied simultaneously.

The policy requirements can also be used to decide whether an individ? ual signature is accepted (= is signed by a recognized key of a known author); in that case some requirements may apply only to some signa? tures, but each signature must be accepted by at least one requirement object.

The following requirement objects are supported:

insecureAcceptAnything

A simple requirement with the following syntax

```
{"type":"insecureAcceptAnything"}
```

This requirement accepts any image (but note that other requirements in the array still apply).

When deciding to accept an individual signature, this requirement does not have any effect; it does not cause the signature to be accepted, though.

This is useful primarily for policy scopes where no signature verifica? tion is required; because the array of policy requirements must not be empty, this requirement is used to represent the lack of requirements explicitly.

reject

A simple requirement with the following syntax:

```
{"type":"reject"}
```

This requirement rejects every image, and every signature.

signedBy

This requirement requires an image to be signed using ?simple signing?
with an expected identity, or accepts a signature if it is using an ex?
pected identity and key.

```
       {
           "type":    "signedBy",
           "keyType": "GPGKeys", /* The only currently supported value */
           "keyPath": "/path/to/local/keyring/file",
           "keyPaths": ["/path/to/local/keyring/file1","/path/to/local/keyring/file2"?],
           "keyData": "base64-encoded-keyring-data",
           "signedIdentity": identity_requirement
       }
```

Exactly one of keyPath, keyPaths and keyData must be present, contain?
ing a GPG keyring of one or more public keys. Only signatures made by
these keys are accepted.

The signedIdentity field, a JSON object, specifies what image identity
the signature claims about the image. One of the following alterna?
tives are supported:

? The identity in the signature must exactly match the image
  identity. Note that with this, referencing an image by digest
  (with a signature claiming a repository:tag identity) will
  fail.

  {"type":"matchExact"}

? If the image identity carries a tag, the identity in the sig?
  nature must exactly match; if the image identity uses a digest
  reference, the identity in the signature must be in the same
  repository as the image identity (using any tag).

(Note that with images identified using digest references, the digest
from the reference is validated even before signature verification
starts.)

  {"type":"matchRepoDigestOrExact"}

? The identity in the signature must be in the same repository
  as the image identity. This is useful e.g. to pull an image
  using the :latest tag when the image is signed with a tag

specifying an exact image version.

{"type":"matchRepository"}

? The identity in the signature must exactly match a specified

identity.   This is useful e.g. when locally mirroring images

signed using their public identity.

```
{
    "type": "exactReference",
    "dockerReference": docker_reference_value
}
```

? The identity in the signature must be in the same repository

as a specified identity.   This combines the properties of

matchRepository and exactReference.

```
{
    "type": "exactRepository",
    "dockerRepository": docker_repository_value
}
```

? Prefix remapping:

If the image identity matches the specified prefix, that prefix is re?

placed by the specified ?signed prefix?

(otherwise it is used as unchanged and no remapping takes place);

matching then follows the matchRepoDigestOrExact semantics documented

above

(i.e. if the image identity carries a tag, the identity in the signa?

ture must exactly match,

if it uses a digest reference, the repository must match).

The prefix and signedPrefix values can be either host[:port] values

(matching exactly the same host[:port], string),

repository namespaces, or repositories (i.e. they must not contain

tags/digests),

and match as prefixes of the fully expanded form.

For example, docker.io/library/busybox (not busybox) to specify that

single repository,

or docker.io/library (not an empty string) to specify the parent namespace of docker.io/library/busybox==busybox).

The prefix value is usually the same as the scope containing the parent signedBy requirement.

```
  {
     "type": "remapIdentity",
     "prefix": prefix,
     "signedPrefix": prefix,
  }
```

If the signedIdentity field is missing, it is treated as matchRepoDige?stOrExact.

Note: matchExact, matchRepoDigestOrExact and matchRepository can be only used if a Docker-like image identity is provided by the transport. In particular, the dir: and oci: transports can be only used with exac?tReference or exactRepository.

sigstoreSigned

This requirement requires an image to be signed using a sigstore signa?ture with an expected identity and key.

```
{
   "type":    "sigstoreSigned",
   "keyPath": "/path/to/local/public/key/file",
   "keyData": "base64-encoded-public-key-data",
   "fulcio": {
      "caPath": "/path/to/local/CA/file",
      "caData": "base64-encoded-CA-data",
      "oidcIssuer": "https://expected.OIDC.issuer/",
      "subjectEmail", "expected-signing-user@example.com",
   },
   "rekorPublicKeyPath": "/path/to/local/public/key/file",
   "rekorPublicKeyData": "base64-encoded-public-key-data",
   "signedIdentity": identity_requirement
}
```

Exactly one of keyPath, keyData and fulcio must be present.

If keyPath or keyData is present, it contains a sigstore public key.

Only signatures made by this key are accepted.

If fulcio is present, the signature must be based on a Fulcio-issued certificate. One of caPath and caData must be specified, containing the public key of the Fulcio instance. Both oidcIssuer and sub?jectEmail are mandatory, exactly specifying the expected identity provider, and the identity of the user obtaining the Fulcio certifi?cate.

At most one of rekorPublicKeyPath and rekorPublicKeyData can be present; it is mandatory if fulcio is specified. If a Rekor public key is specified, the signature must have been uploaded to a Rekor server and the signature must contain an (offline-verifiable) ?signed entry timestamp? proving the existence of the Rekor log record, signed by the provided public key.

The signedIdentity field has the same semantics as in the signedBy re?quirement described above. Note that cosign-created signatures only contain a repository, so only matchRepository and exactRepository can be used to accept them (and that does not protect against substitution of a signed image with an unexpected tag).

To use this with images hosted on image registries, the relevant reg?istry or repository must have the use-sigstore-attachments option en?abled in containers-registries.d(5).

Examples

It is strongly recommended to set the default policy to reject, and then selectively allow individual transports and scopes as desired.

A reasonably locked-down system

(Note that the /*?*/ comments are not valid in JSON, and must not be used in real policies.)

```
{
    "default": [{"type": "reject"}], /* Reject anything not explicitly allowed */
    "transports": {
        "docker": {
            /* Allow installing images from a specific repository namespace, without cryptographic verification.
               This namespace includes images like openshift/hello-openshift and openshift/origin. */
```

```
"docker.io/openshift": [{"type": "insecureAcceptAnything"}],
/* Similarly, allow installing the ?official? busybox images.  Note how the fully expanded
   form, with the explicit /library/, must be used. */
"docker.io/library/busybox": [{"type": "insecureAcceptAnything"}],
/* Allow installing images from all subdomains */
"*.temporary-project.example.com": [{"type": "insecureAcceptAnything"}],
/* A sigstore-signed repository */
"hostname:5000/myns/sigstore-signed-with-full-references": [
    {
        "type": "sigstoreSigned",
        "keyPath": "/path/to/sigstore-pubkey.pub"
    }
],
/* A sigstore-signed repository using the community Fulcio+Rekor servers.
   The community servers? public keys can be obtained from
   https://github.com/sigstore/sigstore/tree/main/pkg/tuf/repository/targets .  */
"hostname:5000/myns/sigstore-signed-fulcio-rekor": [
    {
        "type": "sigstoreSigned",
        "fulcio": {
            "caPath": "/path/to/fulcio_v1.crt.pem",
            "oidcIssuer": "https://github.com/login/oauth",
            "subjectEmail": "test-user@example.com"
        },
        "rekorPublicKeyPath": "/path/to/rekor.pub",
    }
],
/* A sigstore-signed repository, accepts signatures by /usr/bin/cosign */
"hostname:5000/myns/sigstore-signed-allows-malicious-tag-substitution": [
    {
        "type": "sigstoreSigned",
        "keyPath": "/path/to/sigstore-pubkey.pub",
        "signedIdentity": {"type": "matchRepository"}
        }
    ],
    /* A sigstore-signed repository using the community Fulcio+Rekor servers,
       accepts signatures by /usr/bin/cosign.
       The community servers? public keys can be obtained from
       https://github.com/sigstore/sigstore/tree/main/pkg/tuf/repository/targets .  */
    "hostname:5000/myns/sigstore-signed-fulcio-rekor- allows-malicious-tag-substitution": [
        {
            "type": "sigstoreSigned",
            "fulcio": {
                "caPath": "/path/to/fulcio_v1.crt.pem",
                "oidcIssuer": "https://github.com/login/oauth",
                "subjectEmail": "test-user@example.com"
            },
            "rekorPublicKeyPath": "/path/to/rekor.pub",
            "signedIdentity": { "type": "matchRepository" }
        }
    ]
      /* Other docker: images use the global default policy and are rejected */
},
"dir": {
    "": [{"type": "insecureAcceptAnything"}] /* Allow any images originating in local directories */
},
"atomic": {
    /* The common case: using a known key for a repository or set of repositories */
    "hostname:5000/myns/official": [
        {
            "type": "signedBy",
            "keyType": "GPGKeys",
            "keyPath": "/path/to/official-pubkey.gpg"
        }
    ],
    /* A more complex example, for a repository which contains a mirror of a third-party product,
          which must be signed-off by local IT */
        "hostname:5000/vendor/product": [
            { /* Require the image to be signed by the original vendor, using the vendor's repository location. */
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/path/to/vendor-pubkey.gpg",
                "signedIdentity": {
                    "type": "exactRepository",
                    "dockerRepository": "vendor-hostname/product/repository"
                }
            },
            { /* Require the image to _also_ be signed by a local reviewer. */
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/path/to/reviewer-pubkey.gpg"
            }
        ],
        /* A way to mirror many repositories from a single vendor */
        "private-mirror:5000/vendor-mirror": [
            { /* Require the image to be signed by the original vendor, using the vendor's repository location.
                 For example, private-mirror:5000/vendor-mirror/productA/image1:latest needs to be signed as
                 vendor.example/productA/image1:latest . */
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/path/to/vendor-pubkey.gpg",
                "signedIdentity": {
                    "type": "remapIdentity",
                    "prefix": "private-mirror:5000/vendor-mirror",
                    "signedPrefix": "vendor.example.com"
                }
            }
        ]
    }
```

```
            }
        }
```

Completely disable security, allow all images, do not trust any signatures

```
        {
            "default": [{"type": "insecureAcceptAnything"}]
        }
```

SEE ALSO

atomic(1)

HISTORY

August 2018, Rename to containers-policy.json(5) by Valentin Rothberg

vrothberg@suse.com ?mailto:vrothberg@suse.com?

September 2016, Originally compiled by Miloslav Trma? mitr@redhat.com

?mailto:mitr@redhat.com?