Confidential ACI

Security Policy
aka Execution Policy

Ken Gordon (at microsoft.com)

Many slides by Matthew Johnson

Practical, deployed, integrated into Azure, but also flexible.

Confidential containers on Azure Container Instances - Azure Container Instances | Microsoft Learn

microsoft/hcsshim: Windows - Host Compute Service Shim (github.com)

microsoft/confidential-aci-examples: A collection of examples and tests to run on Confidential Azure Container Instances (github.com)

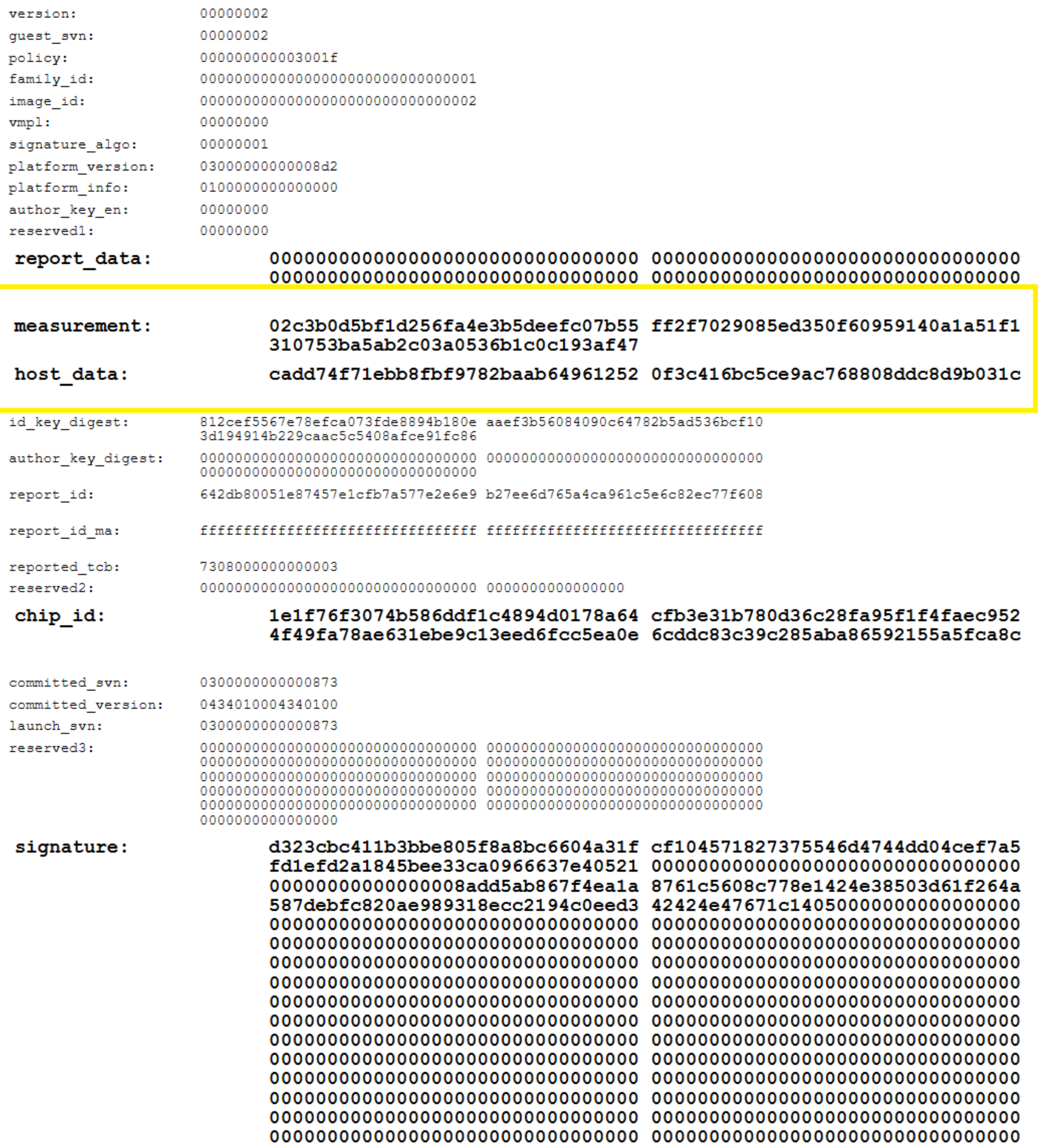
LCOW

- Confidential ACI provides serverless Confidential Containers in Azure using LCOW (Linux Containers on Windows).
- Pods (aka Container Groups) owned by a customer are run in a Hyper-V isolated VM (UVM) solely for use by that customer. This VM is encrypted and protected against the host using AMD SEV-SNP.
- Containerd runs on the (Windows) host and fetches regular container images.

Delta for confidential

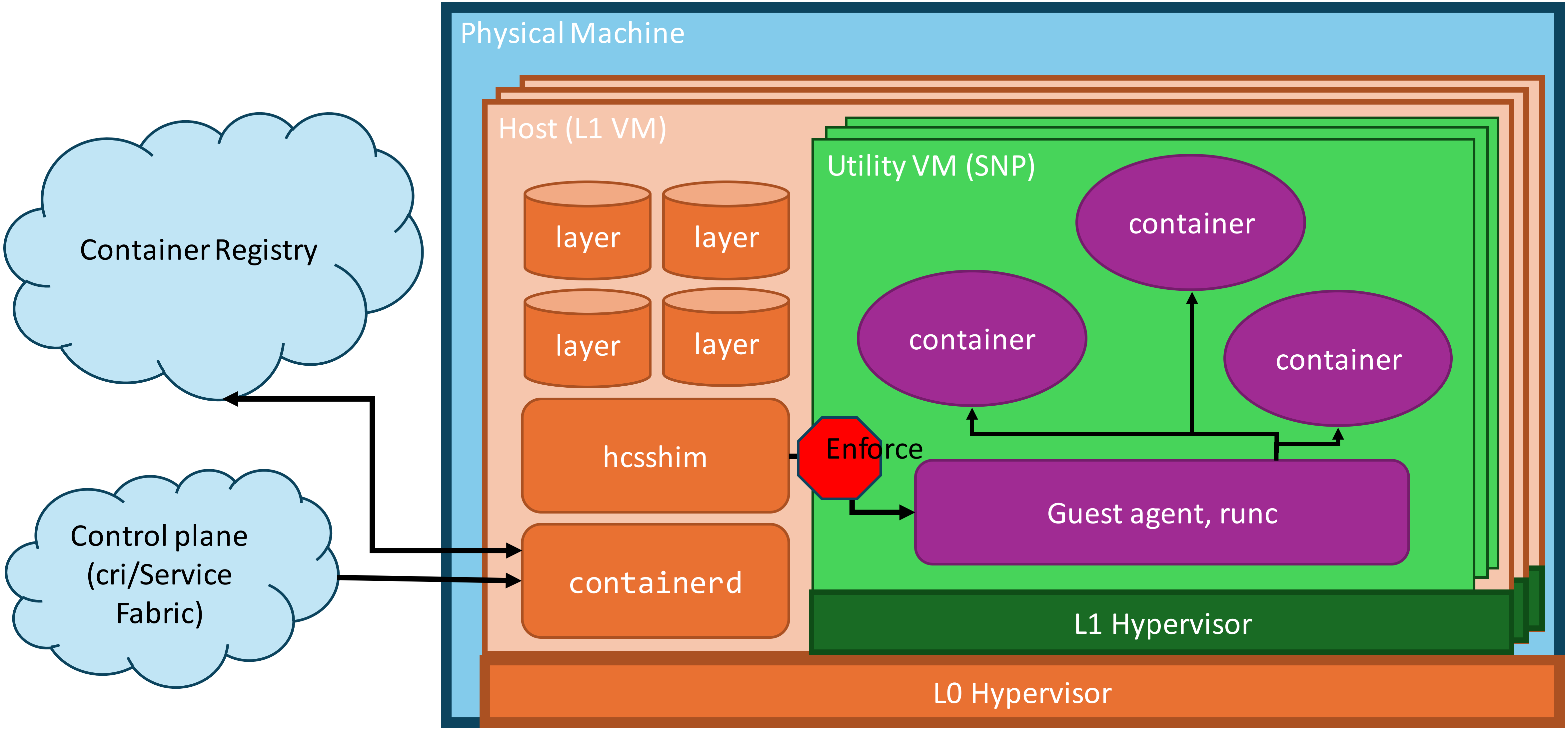
- The Linux guest OS (in the UVM) is minimalist and only runs the guest agent.
- An execution policy (in rego) controls what the host/control plane can ask the UVM to do.
- The guest OS is measured by the PSP and the measurement and hash of the policy is in the hardware attestation reports.
- Image layers have a Merkle tree, regenerated locally.
- “Scratch space” VHDs used for RW parts is encrypted.
- Most users will use tooling to generate the policy. Eg az confcom | Microsoft Learn

Payload generated AMD SEV-SNP Attestation Report:

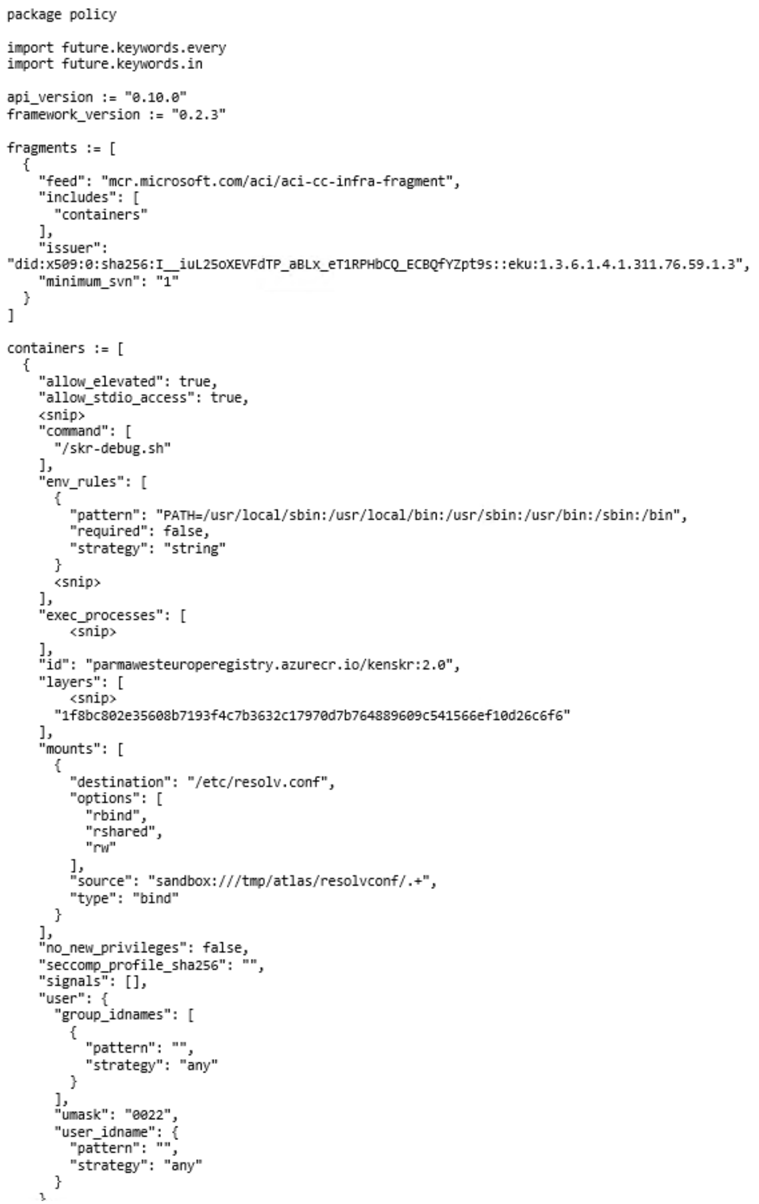


```
version:             00000002
guest_svn:           00000002
policy:              000000000003001f
family_id:           00000000000000000000000000000001
image_id:            00000000000000000000000000000002
vmpl:                00000000
signature_algo:      00000001
platform_version:    03000000000008d2
platform_info:       0100000000000000
author_key_en:       00000000
reserved1:           00000000
report_data:         00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000

measurement:         02c3b0d5bf1d256fa4e3b5deefc07b55 ff2f7029085ed350f60959140a1a51f1
                     310753ba5ab2c03a0536b1c0c193af47

host_data:           cadd74f71ebb8fbf9782baab64961252 0f3c416bc5ce9ac768808ddc8d9b031c

id_key_digest:       812cef5567e78efca073fde8894b180e aaef3b56084090c64782b5ad536bcf10
                     3d194914b229caac5c5408afce91fc86

author_key_digest:   00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000

report_id:           642db80051e87457e1cfb7a577e2e6e9 b27ee6d765a4ca961c5e6c82ec77f608

report_id_ma:        ffffffffffffffffffffffffffffffff ffffffffffffffffffffffffffffffff

reported_tcb:        7308000000000003
reserved2:           00000000000000000000000000000000 0000000000000000
chip_id:             1e1f76f3074b586ddf1c4894d0178a64 cfb3e31b780d36c28fa95f1f4faec952
                     4f49fa78ae631ebe9c13eed6fcc5ea0e 6cddc83c39c285aba86592155a5fca8c

committed_svn:       0300000000000873
committed_version:   0434010004340100
launch_svn:          0300000000000873
reserved3:           00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     0000000000000000
signature:           d323cbc411b3bbe805f8a8bc6604a31f cf104571827375546d4744dd04cef7a5
                     fd1efd2a1845bee33ca0966637e40521 00000000000000000000000000000000
                     00000000000000008add5ab867f4ea1a 8761c5608c778e1424e38503d61f264a
                     587debfc820ae989318ecc2194c0eed3 42424e47671c14050000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
                     00000000000000000000000000000000 00000000000000000000000000000000
```

LCOW Infrastructure in Azure



A quick example to set the scene.



```
package policy

import future.keywords.every
import future.keywords.in

api_version := "0.10.0"
framework_version := "0.2.3"

fragments := [
  {
    "feed": "mcr.microsoft.com/aci/aci-cc-infra-fragment",
    "includes": [
      "containers"
    ],
    "issuer":
"did:x509:0:sha256:I__iuL25oXEVFdTP_aBLx_eT1RPHbCQ_ECBQfYZpt9s::eku:1.3.6.1.4.1.311.76.59.1.3",
    "minimum_svn": "1"
  }
]

containers := [
  {
    "allow_elevated": true,
    "allow_stdio_access": true,
    <snip>
    "command": [
      "/skr-debug.sh"
    ],
    "env_rules": [
      {
        "pattern": "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "required": false,
        "strategy": "string"
      }
      <snip>
    ],
    "exec_processes": [
        <snip>
    ],
    "id": "parmawesteuroperegistry.azurecr.io/kenskr:2.0",
    "layers": [
        <snip>
      "1f8bc802e35608b7193f4c7b3632c17970d7b764889609c541566ef10d26c6f6"
    ],
    "mounts": [
      {
        "destination": "/etc/resolv.conf",
        "options": [
          "rbind",
          "rshared",
          "rw"
        ],
        "source": "sandbox:///tmp/atlas/resolvconf/.+",
        "type": "bind"
      }
    ],
    "no_new_privileges": false,
    "seccomp_profile_sha256": "",
    "signals": [],
    "user": {
      "group_idnames": [
        {
          "pattern": "",
          "strategy": "any"
        }
      ],
      "umask": "0022",
      "user_idname": {
        "pattern": "",
        "strategy": "any"
      }
    }
```

What did policy ever do for Confidential Computing?

Necessary features to ensure a workload is confidential

Protect sensitive memory - encryption/isolation

Be sure we are running in a genuine environment - attestation

Prevent interference - integrity

Control what the environment can do – policy

Release secret gated on complying with the above – relying party

How did we live without it?

Types of Policy:

Implicit policy – attested code can only do what the code can do – e.g. SGX

Explicit policy – attested code uses rules to control its behaviour

Configuration – data which is policy really, eg “only run the next blob if the blob’s hash is X”

Here we are talking about explicit policy expressed as Rego for Confidential Containers “Security Policy”

Where is the magic?

Confidential Containers run inside an encrypted UVM (Utility VM) via an agent

Before UVM start HOST_DATA (an immutable hardware register) is set to sha256(policy)
Next the UVM is measured by AMD hardware
Then it starts, is given the policy and checks the actual policy matches HOST_DATA

In the process of starting containers (and other ops) the agent **enforces the policy**
A container obtains an **attestation report** from the hardware (ignoring key wrapping)
It sends the report to a relying party

The relying party checks that the **UVM measurement** and the **policy hash** it expects are those in the **hardware report**, if so it releases the secret (ignoring mHSM details)

The policy is essential and must prevent unintended changes within the UVM:

- Wrong container
- Wrong container configuration

Attestation.

The workload must provide an attestation report and some collateral to a relying party

The attestation report is a regular AMD SEV-SNP one as illustrated earlier

AMD provides a certificate chain to validate that the attestation report is genuine

Microsoft provides a COSE Sign1 document that allows a relying party to check the hardware UVM measurement is of a genuine image.

Optionally the actual security policy can be provided. For example, CCF will record the whole policy in an immutable log.

The relying party can then check that the UVM measurement is to be trusted (via the COSE Sign1 document or an explicit value) AND that the policy hash (HOST_DATA) is as expected.

Can we explain it?

- Talk about rules or white lists
- Say “allows” about things like container lists
- Make clear the differences

    - Configuration – what ought to be done
    - Policy/rules – what is allowed to be done

- Checking that rules are being respected – correct UVM
- That the correct rules are in place
- The check is for secret release NOT running the code

A UVM cannot check itself is genuine since compromised UVM could run/do anything

Enforcement

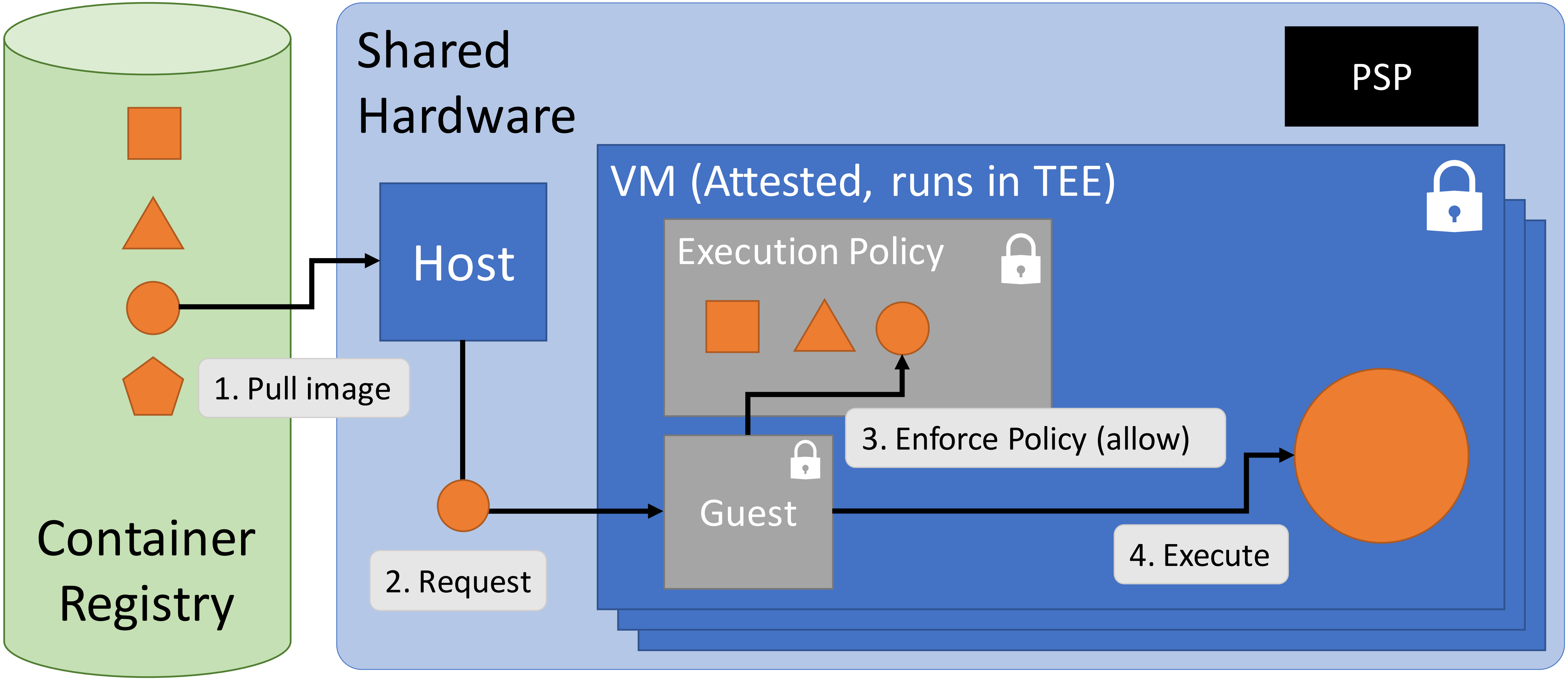
The aim of the policy is to prevent an attacker modifying the state of the UVM or Container to gain access to confidential data

For example, it constrains the containers being loaded to the trusted set specified by the user.

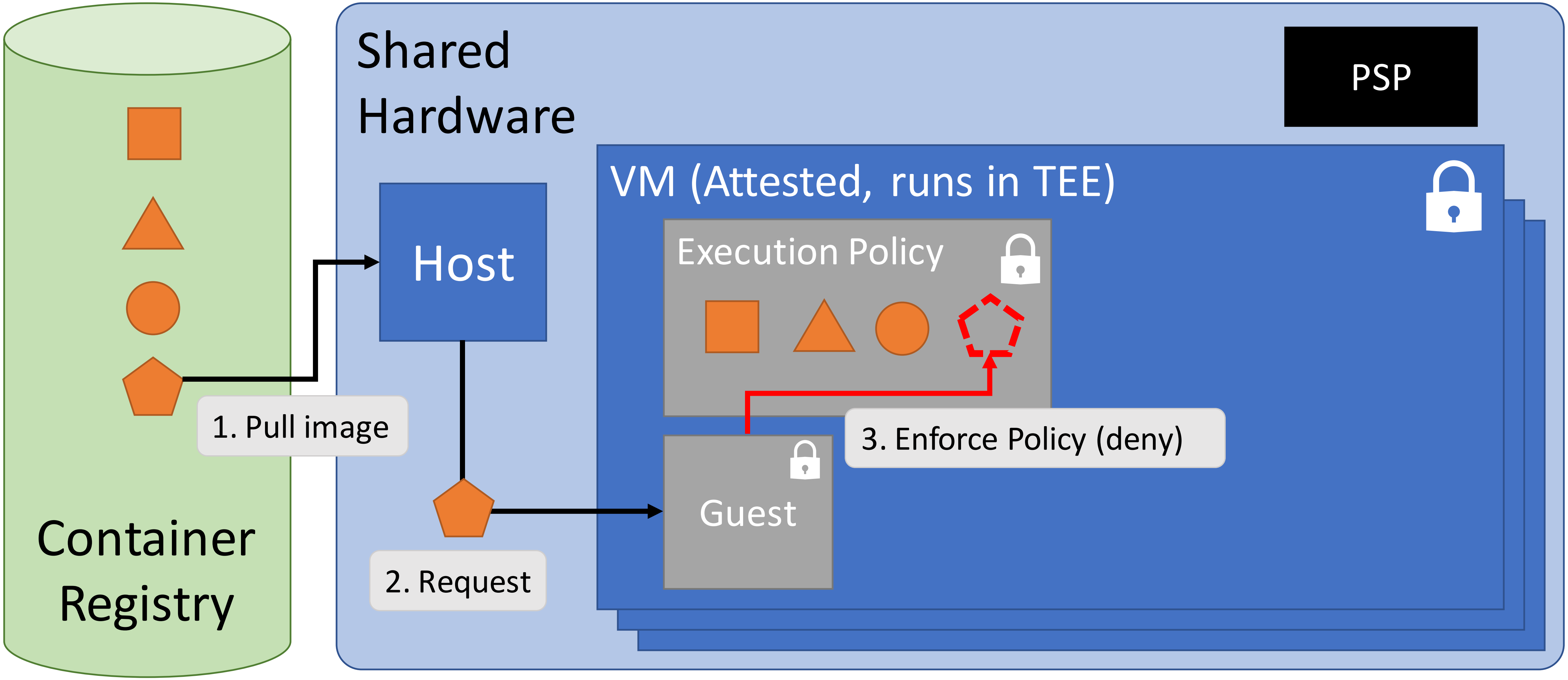
The agent code checks requests against the policy at “enforcement points”
The enforcement points must cover all significant modifications of the system state

Container runtime agents have a narrow API which enables enforcement to be straight-forward

Execution Policy: Allow



Execution Policy: Deny



What should be enforced?

Essentially the VM's external surface.

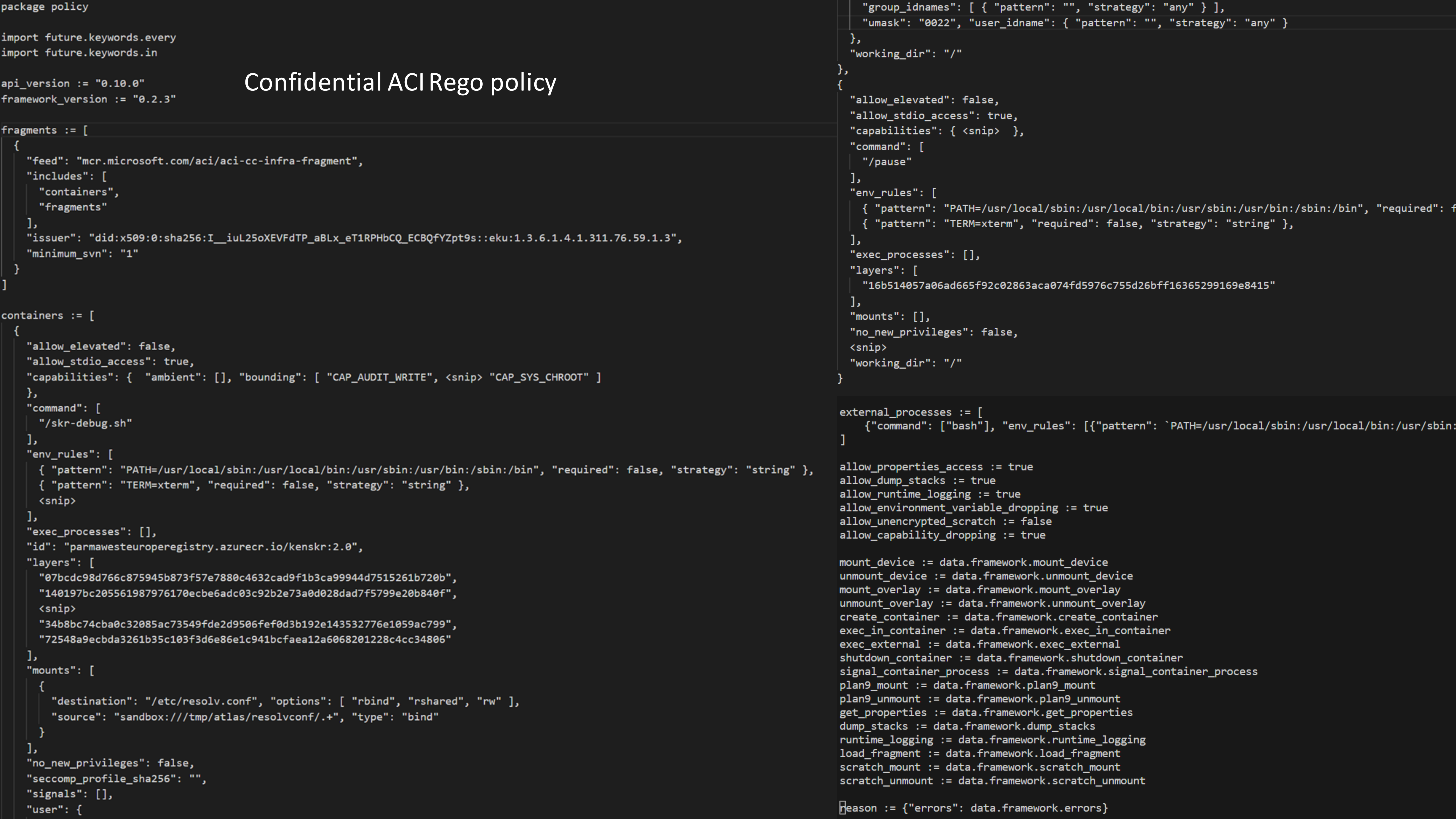
e.g.,
don't let the host mount /evil in place of /bin,

don't allow container evil.acurecr.io/steal_it:latest to load

but expressed as positives

- Mounting devices
- Mounting overlays
- Creating containers
  - Command
  - Environment variables
  - Mounts
  - Processes
  - Logging
  - Signals
  - stdio access

- External (UVM) processes
- External logging
- Plan9 mounts
- Getting properties
- Dumping stacks
- Dropping environment variables
- Dropping capabilities
- Security context
  - RunAsUser, RunAsGroup
  - Capabilities
  - Seccomp
  - NoNewPrivileges



Confidential ACI Rego policy

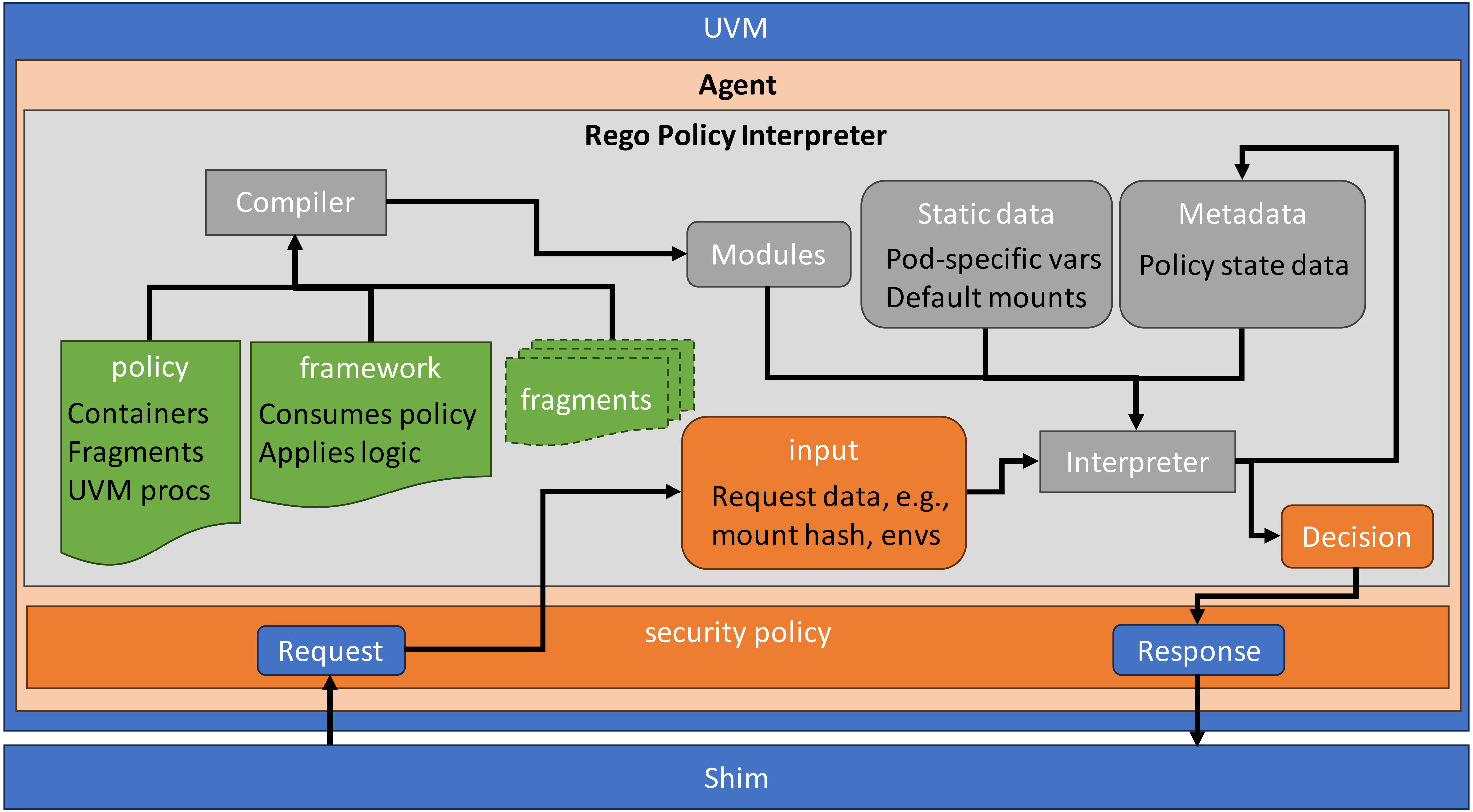
```
package policy

import future.keywords.every
import future.keywords.in

api_version := "0.10.0"
framework_version := "0.2.3"

fragments := [
  {
    "feed": "mcr.microsoft.com/aci/aci-cc-infra-fragment",
    "includes": [
      "containers",
      "fragments"
    ],
    "issuer": "did:x509:0:sha256:I__iuL25oXEVFdTP_aBLx_eT1RPHbCQ_ECBQfYZpt9s::eku:1.3.6.1.4.1.311.76.59.1.3",
    "minimum_svn": "1"
  }
]

containers := [
  {
    "allow_elevated": false,
    "allow_stdio_access": true,
    "capabilities": {  "ambient": [], "bounding": [ "CAP_AUDIT_WRITE", <snip> "CAP_SYS_CHROOT" ]
    },
    "command": [
      "/skr-debug.sh"
    ],
    "env_rules": [
      { "pattern": "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", "required": false, "strategy": "string" },
      { "pattern": "TERM=xterm", "required": false, "strategy": "string" },
      <snip>
    ],
    "exec_processes": [],
    "id": "parmawesteuroperegistry.azurecr.io/kenskr:2.0",
    "layers": [
      "07bcdc98d766c875945b873f57e7880c4632cad9f1b3ca99944d7515261b720b",
      "140197bc205561987976170ecbe6adc03c92b2e73a0d028dad7f5799e20b840f",
      <snip>
      "34b8bc74cba0c32085ac73549fde2d9506fef0d3b192e143532776e1059ac799",
      "72548a9ecbda3261b35c103f3d6e86e1c941bcfaea12a6068201228c4cc34806"
    ],
    "mounts": [
      {
        "destination": "/etc/resolv.conf", "options": [ "rbind", "rshared", "rw" ],
        "source": "sandbox:///tmp/atlas/resolvconf/.+", "type": "bind"
      }
    ],
    "no_new_privileges": false,
    "seccomp_profile_sha256": "",
    "signals": [],
    "user": {
```

```
    "group_idnames": [ { "pattern": "", "strategy": "any" } ],
    "umask": "0022", "user_idname": { "pattern": "", "strategy": "any" }
  },
  "working_dir": "/"
},
{
  "allow_elevated": false,
  "allow_stdio_access": true,
  "capabilities": { <snip>  },
  "command": [
    "/pause"
  ],
  "env_rules": [
    { "pattern": "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin", "required": f
    { "pattern": "TERM=xterm", "required": false, "strategy": "string" },
  ],
  "exec_processes": [],
  "layers": [
    "16b514057a06ad665f92c02863aca074fd5976c755d26bff16365299169e8415"
  ],
  "mounts": [],
  "no_new_privileges": false,
  <snip>
  "working_dir": "/"
}

external_processes := [
    {"command": ["bash"], "env_rules": [{"pattern": `PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:
]

allow_properties_access := true
allow_dump_stacks := true
allow_runtime_logging := true
allow_environment_variable_dropping := true
allow_unencrypted_scratch := false
allow_capability_dropping := true

mount_device := data.framework.mount_device
unmount_device := data.framework.unmount_device
mount_overlay := data.framework.mount_overlay
unmount_overlay := data.framework.unmount_overlay
create_container := data.framework.create_container
exec_in_container := data.framework.exec_in_container
exec_external := data.framework.exec_external
shutdown_container := data.framework.shutdown_container
signal_container_process := data.framework.signal_container_process
plan9_mount := data.framework.plan9_mount
plan9_unmount := data.framework.plan9_unmount
get_properties := data.framework.get_properties
dump_stacks := data.framework.dump_stacks
runtime_logging := data.framework.runtime_logging
load_fragment := data.framework.load_fragment
scratch_mount := data.framework.scratch_mount
scratch_unmount := data.framework.scratch_unmount

reason := {"errors": data.framework.errors}
```

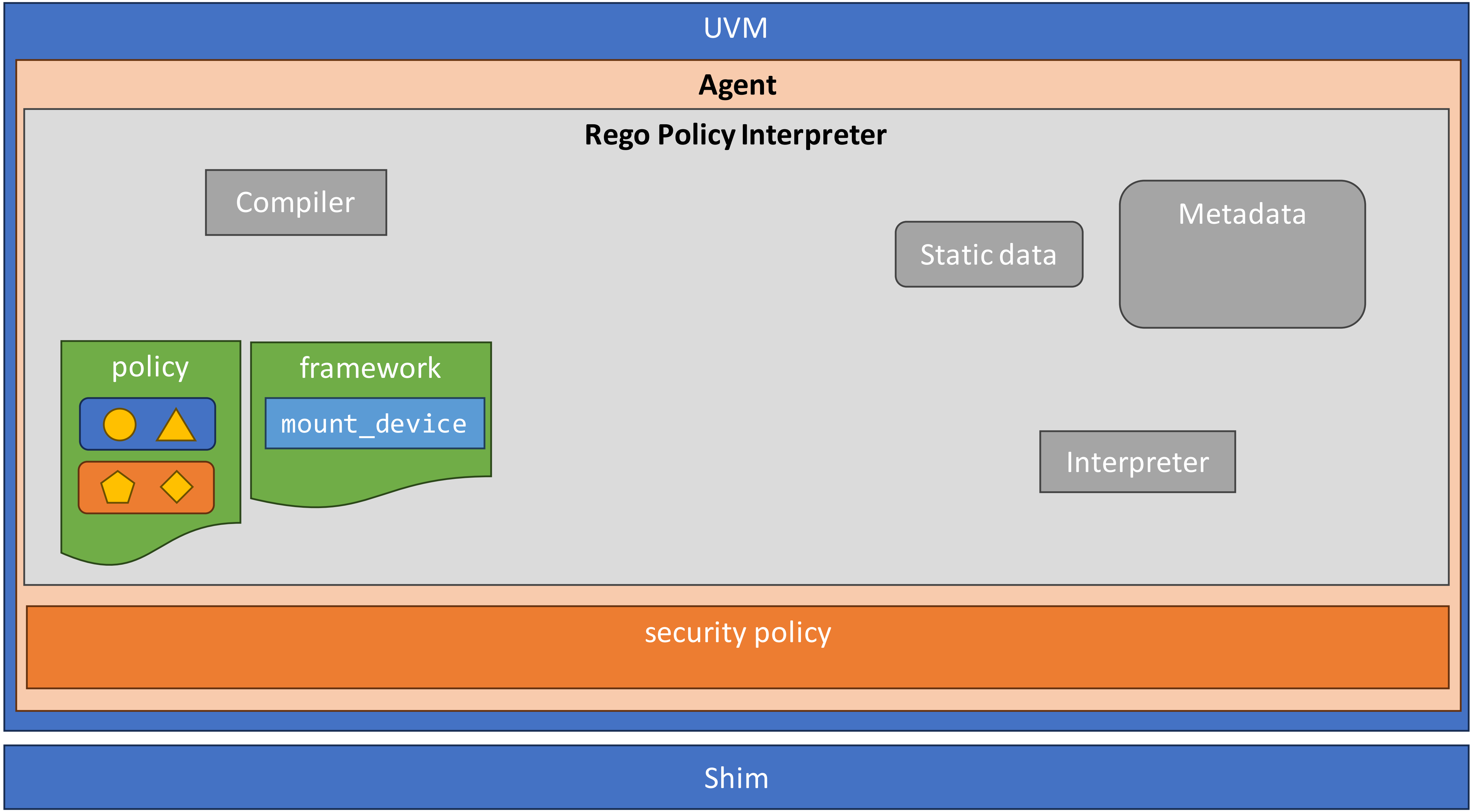
Execution Policy Engine in Rego
Framework code operates over that user policy

```
default mount_device := {"allowed": false}

mount_device := {"allowed": true} {
    some container in data.policy.containers
    some layer in container.layers
    input.deviceHash == layer
}
```

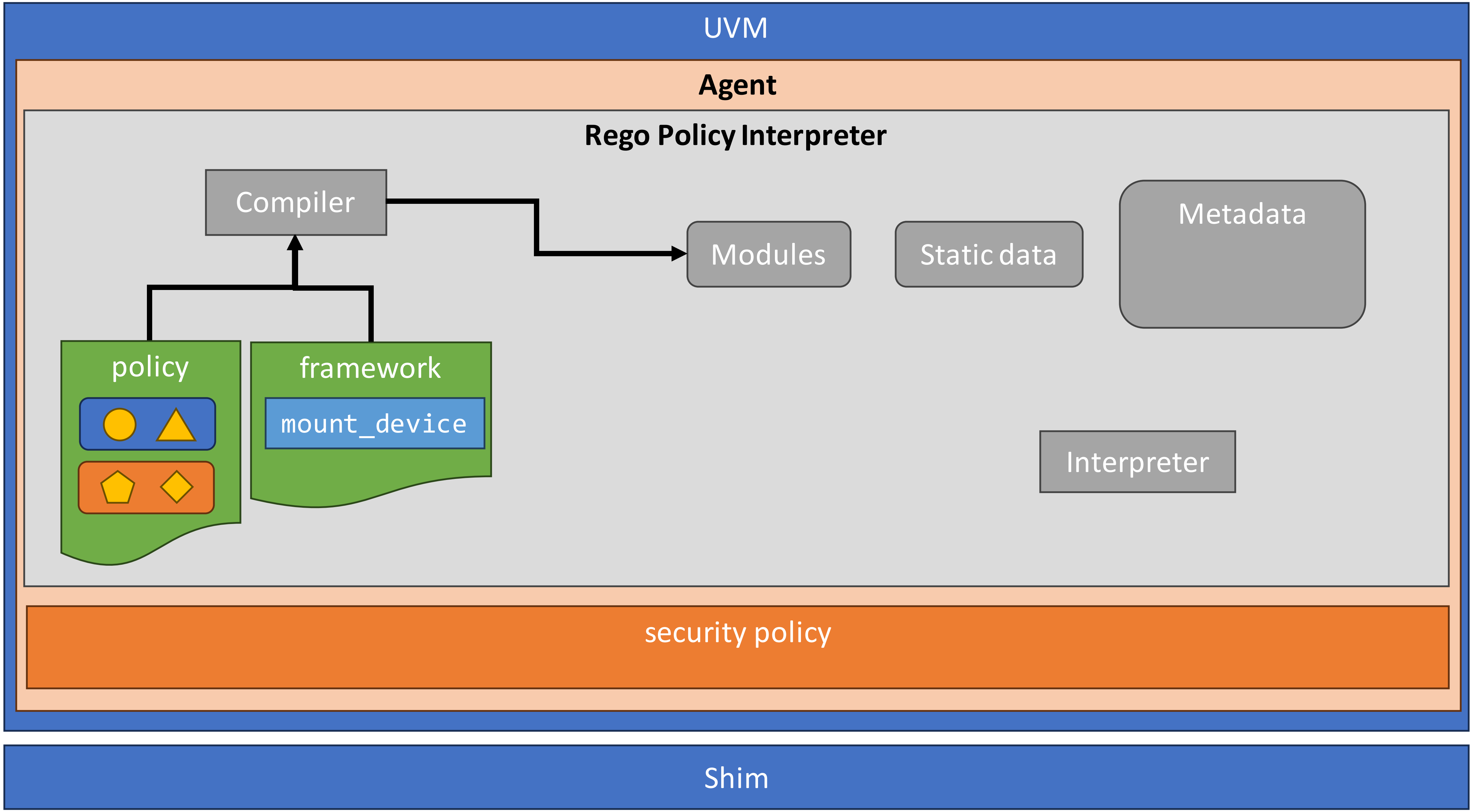
Confidential ACI – implementation overview



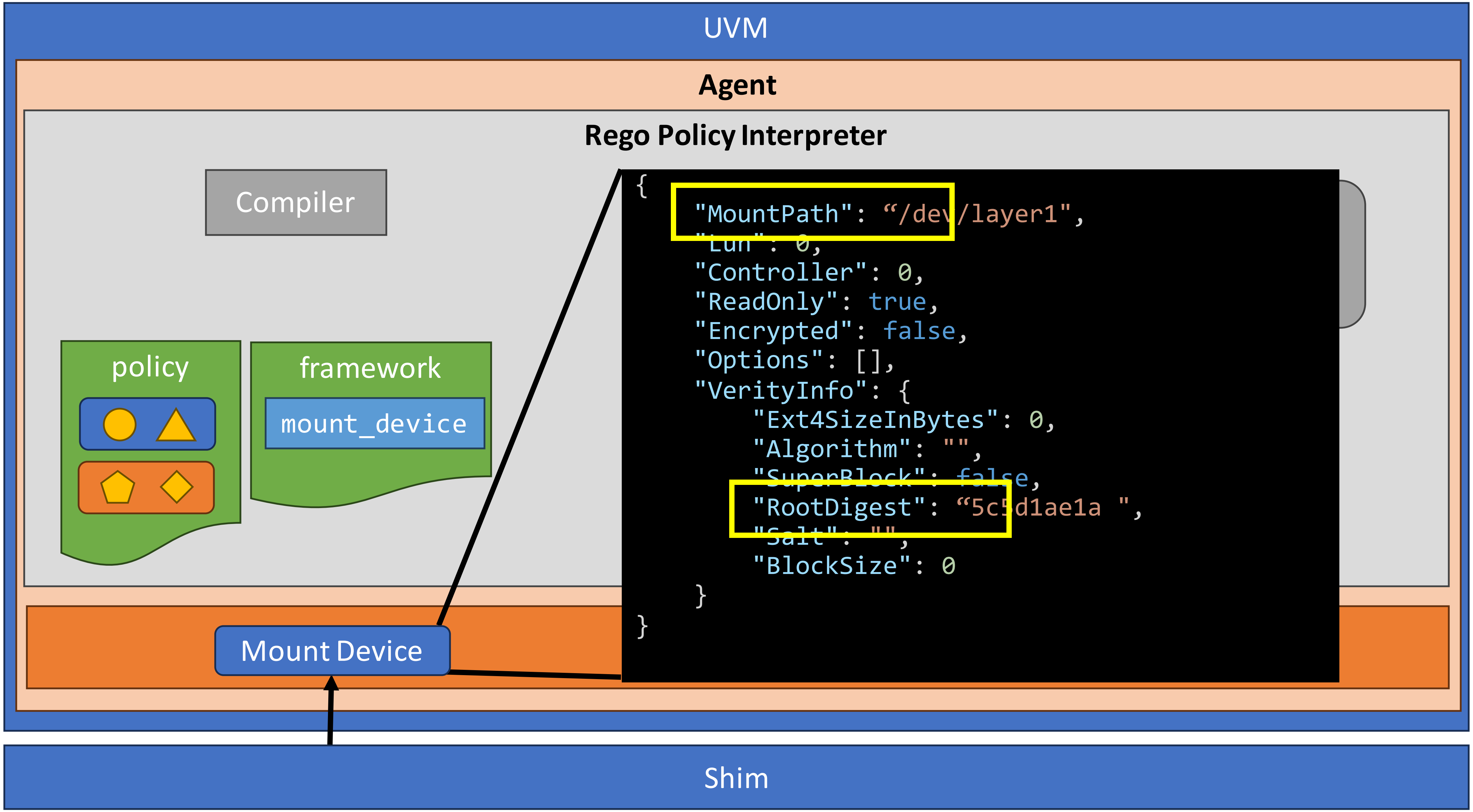
The shim provides the security policy, gcs provides the framework.



They get compiled into rego modules

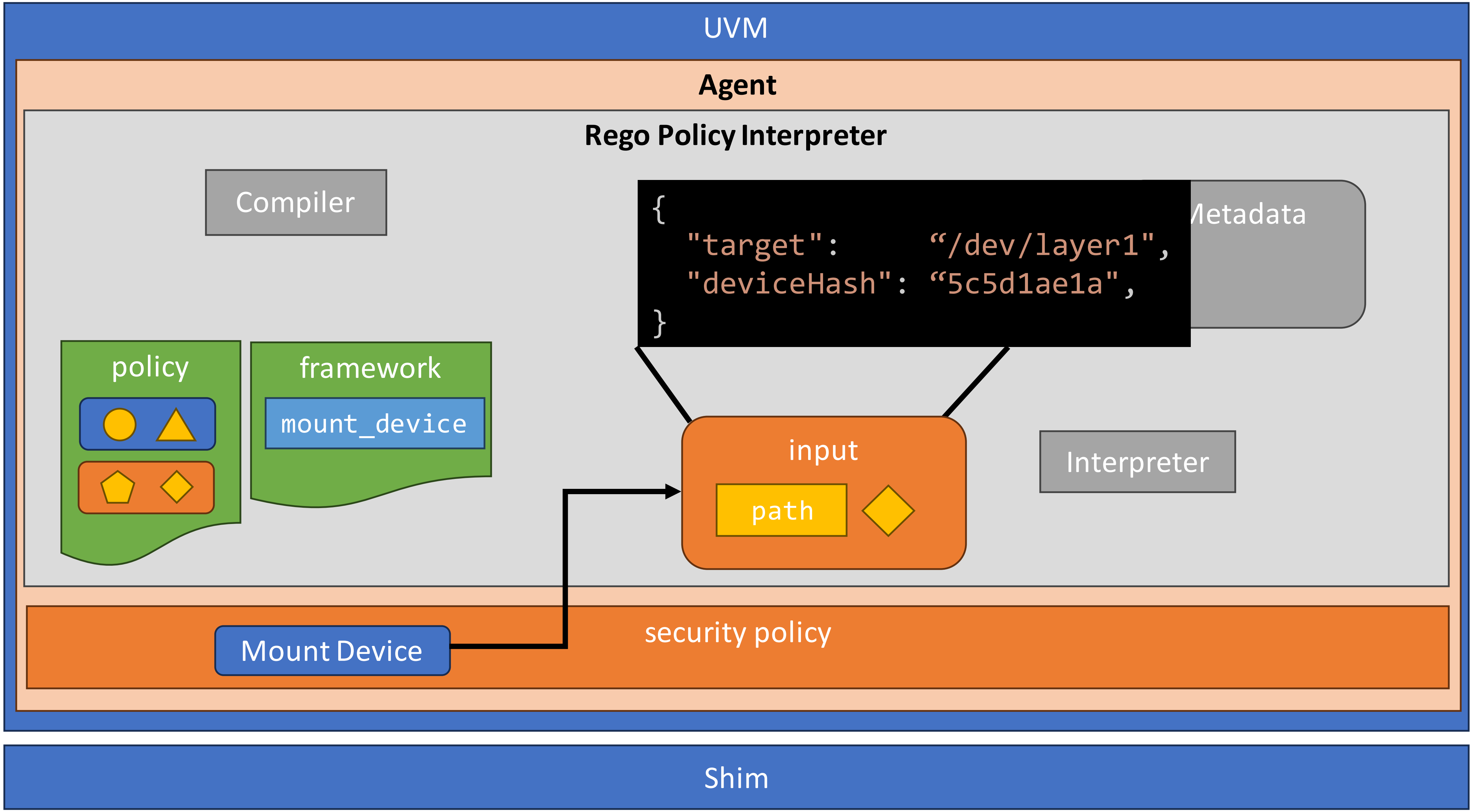


A request comes from the shim and is presented to the go security policy code



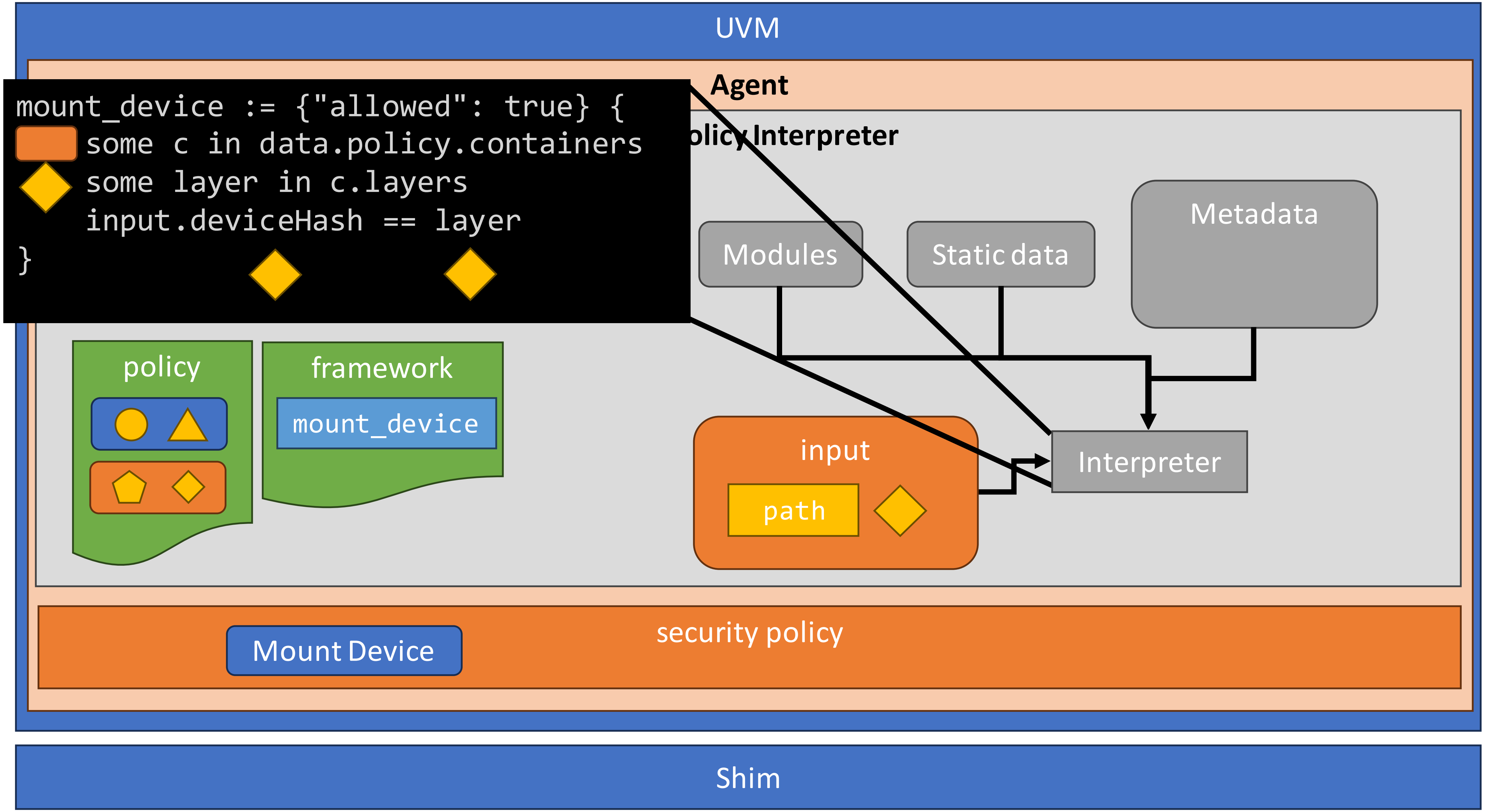
```
{
    "MountPath": “/dev/layer1",
    "Lun": 0,
    "Controller": 0,
    "ReadOnly": true,
    "Encrypted": false,
    "Options": [],
    "VerityInfo": {
        "Ext4SizeInBytes": 0,
        "Algorithm": "",
        "SuperBlock": false,
        "RootDigest": “5c5d1ae1a ",
        "Salt": "",
        "BlockSize": 0
    }
}
```

An “input” object is presented to the framework



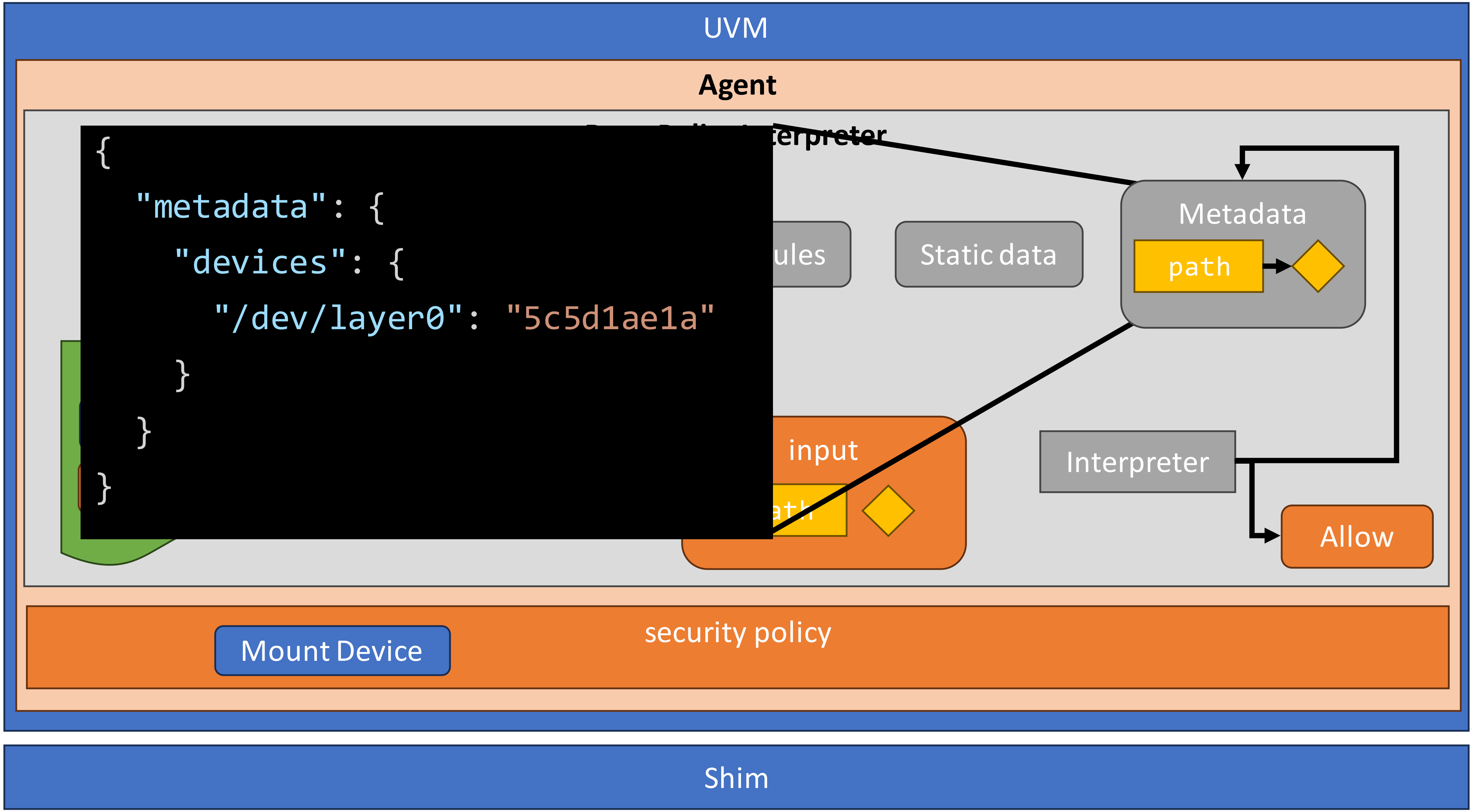
```
{
  "target":     “/dev/layer1",
  "deviceHash": “5c5d1ae1a",
}
```

Rules in the framework test the request



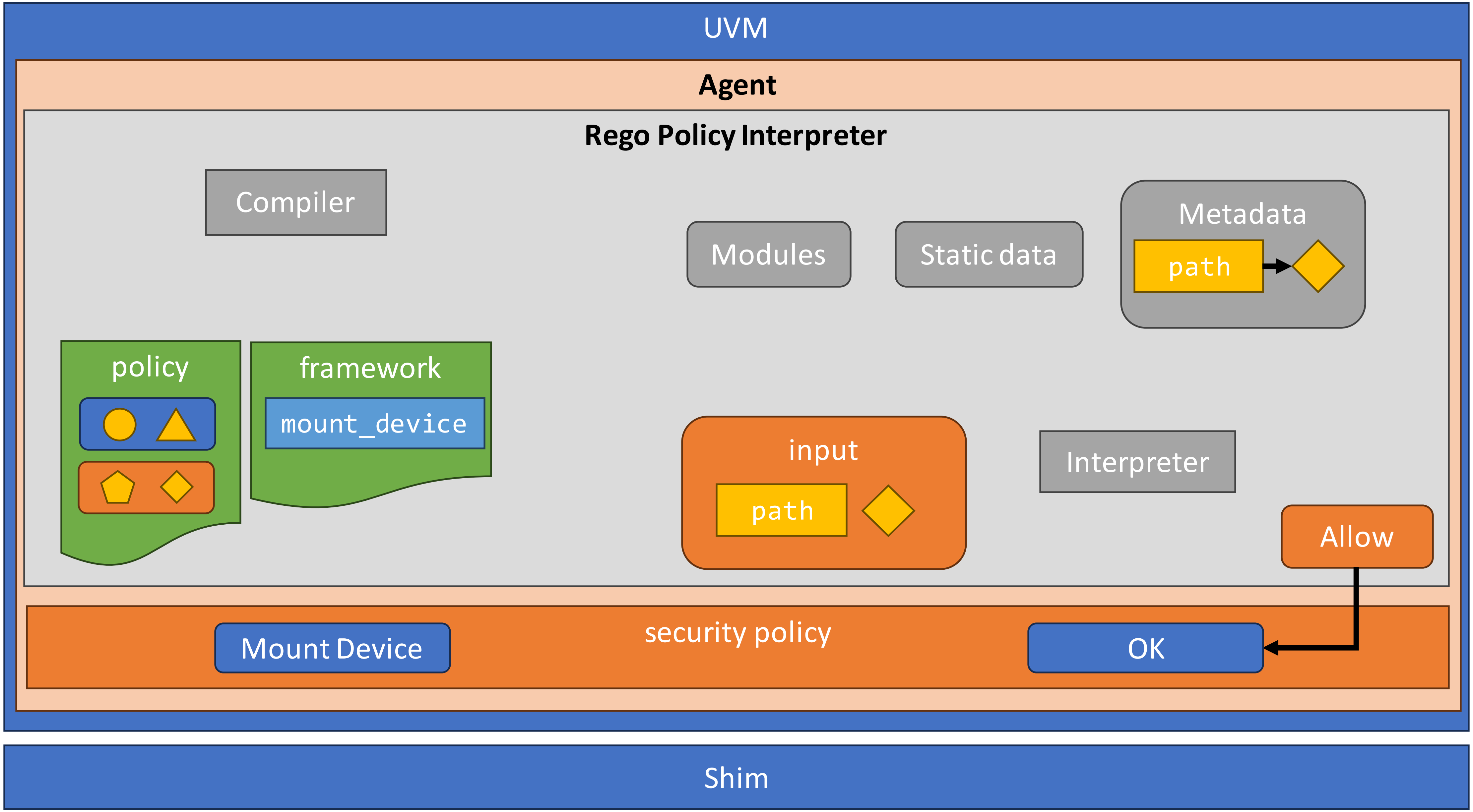
```
mount_device := {"allowed": true} {
🟧 some c in data.policy.containers
◆  some layer in c.layers
   input.deviceHash == layer
}
```

If the test passes “metadata” state may be captured (narrowing etc)



```
{
  "metadata": {
    "devices": {
      "/dev/layer0": "5c5d1ae1a"
    }
  }
}
```

If the test passed then “Allow” is returned to the go code



Managing change

A real cloud, real customers, no pontification and baseless assertions.

User containers change, infrastructure changes, Linux gets updated….

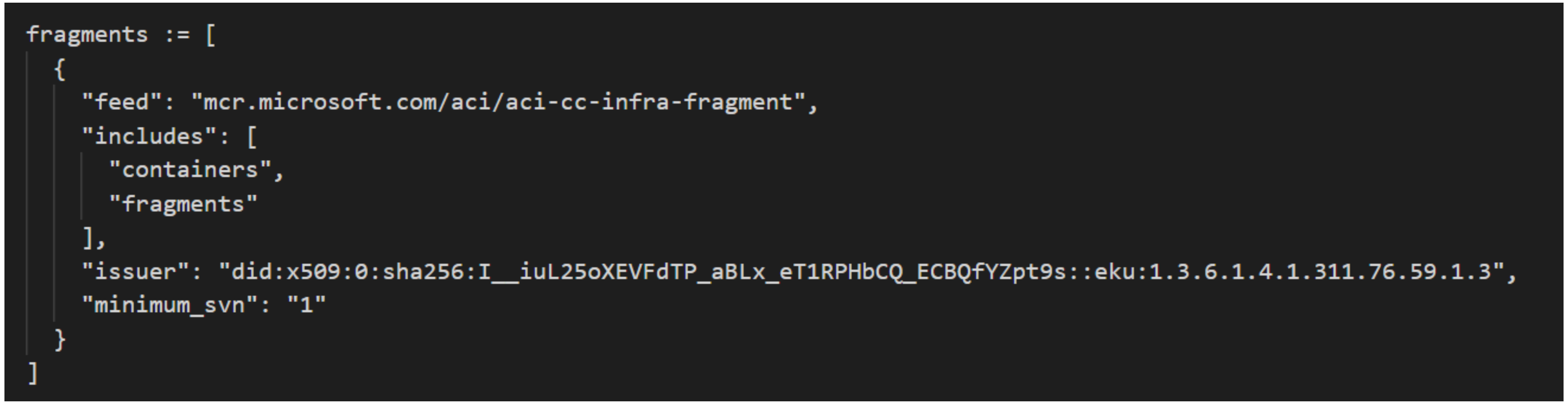
Change is everywhere and unavoidable.

Worse still, it is asynchronous, stuff goes forwards and backwards. How do we stop it going sideways?

We want the key release policy/relying party config to be stable in the face of approved change.

Fragments

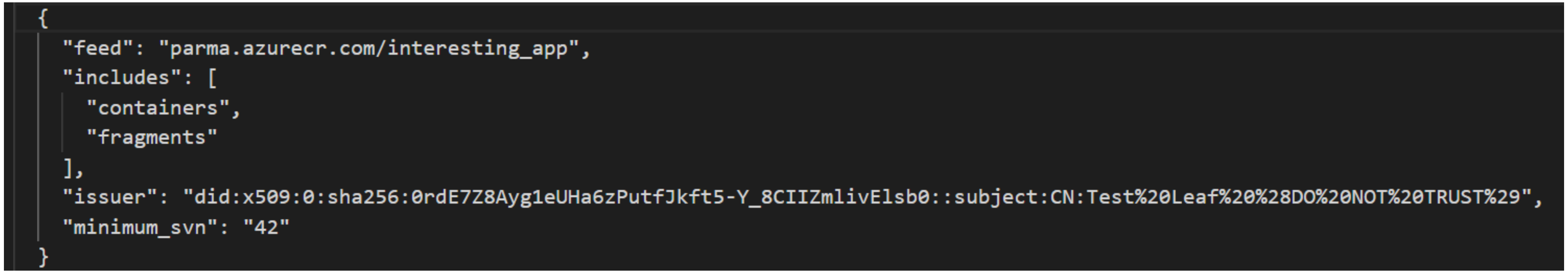
- A piece of policy outside of the user supplied policy but allowed in by that policy
- A rule says “a fragment with an issuer DID of X (ie signed by a trusted party) and feed of Y
- Enables serviceability – this example allows ACI to update infra sidecars



```
fragments := [
  {
    "feed": "mcr.microsoft.com/aci/aci-cc-infra-fragment",
    "includes": [
      "containers",
      "fragments"
    ],
    "issuer": "did:x509:0:sha256:I__iuL25oXEVFdTP_aBLx_eT1RPHbCQ_ECBQfYZpt9s::eku:1.3.6.1.4.1.311.76.59.1.3",
    "minimum_svn": "1"
  }
]
```

Image Attached Fragments (real soon now...)

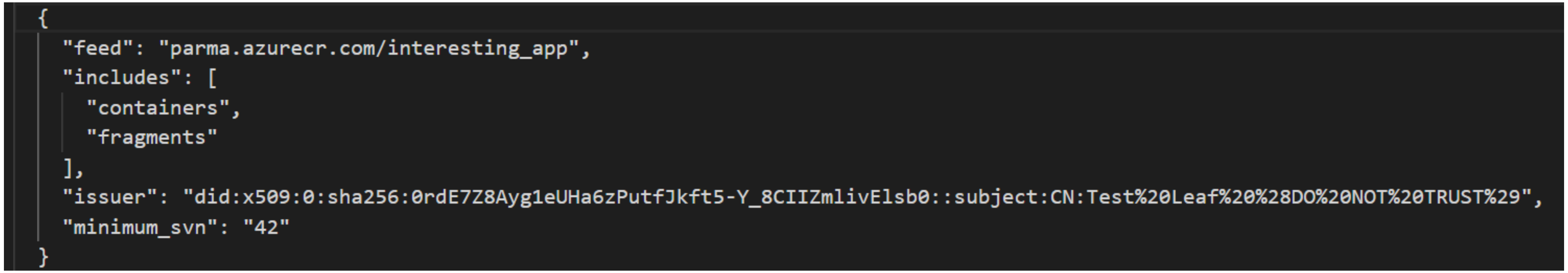
- A fragment which may come from the same container repository as a loaded container (using ORAS)
- Enables CUSTOMER serviceability – this example allows Parma to update interesting_app at will – so long as they can sign the fragment
- Enables stable policy (required for practical key management) in the face of realistic customer change, especially if they have thousands of containers



```
{
  "feed": "parma.azurecr.com/interesting_app",
  "includes": [
    "containers",
    "fragments"
  ],
  "issuer": "did:x509:0:sha256:0rdE7Z8Ayg1eUHa6zPutfJkft5-Y_8CIIZmlivElsb0::subject:CN:Test%20Leaf%20%28DO%20NOT%20TRUST%29",
  "minimum_svn": "42"
}
```

Standalone Fragments (future feature)

- A fragment which can come from the any container repository (using ORAS) – not specific to a container – path must be provided via arm template
- Allows customer to completely abstract away the policy
- The customer version of the ACI infra fragment



```
{
  "feed": "parma.azurecr.com/interesting_app",
  "includes": [
    "containers",
    "fragments"
  ],
  "issuer": "did:x509:0:sha256:0rdE7Z8Ayg1eUHa6zPutfJkft5-Y_8CIIZmlivElsb0::subject:CN:Test%20Leaf%20%28DO%20NOT%20TRUST%29",
  "minimum_svn": "42"
}
```

DID:x509 and COSE_Sign1

- microsoft/did-x509: DRAFT: did:x509 Decentralized Identifier Method Specification (github.com)
- https://github.com/microsoft/didx509go

- CBOR Object Signing and Encryption (COSE) (cose-wg.github.io)
- microsoft/cosesign1go: A Go library to handle COSE Sign1 documents (github.com)

Questions?

Spare/work in progress slides…

Trusting the platform

- The UVM is measured by the PSP into LAUNCH_MEASUREMENT
- A COSE Sign1 document, signed by Microsoft, asserts a given measurement is genuine.
- Given the UVM is genuine, we know it will properly enforce the supplied execution policy.
- A hash of the execution policy is bond to the hardware via the HOST_DATA field.
- The LAUNCH_MEASUREMENT and HOST_DATA are included in attestation reports generated by the PSP. That supported by a certificate chain from AMD.

```
./sign1util.exe print -in LinuxBootFiles/reference_info.cose
checkCoseSign1 passed
iss: did:x509:0:sha256:I__iuL25oXEVFdTP_aBLx_eT1RPHbCQ_ECBQfYZpt9s::eku:1.3.6.1.4.1.311.76.59.1.2
feed: ContainerPlat-AMD-UVM
cty: application/json
pubkey:
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEAtczxdBvjfCVvCevjsNBcts5qe/y+dG1cTXD8Ge2XNkNA91gwkxvQB2NoQMFmXRkvpH7D/zvhfa1Le110YdzSFykKN4eRwh8UZfLH
sIVo/JjE7z4gHC6ZX5HONW8Y2eow9Zx5UWb40SKoyj+LMCE6srhCyxb/93RYBTKER7ndtdDwhgq0OQtpFRwjt0ThTtURzRMKDsAeGoaex+Kn5cVuXz3CrX6AB3RBDnEg7D8QHnCRjWWR
5hldeZCfMqmbBjQwgcyfHVyotpb81BCNbAtc0K1Nix5HB8lu+b2XyO8vBiEkA/v6/JA00gg/I1MiF++UB5DYbXGYk2POyQHLe80E0BUDogvlszcr0WaoRovi5iUJhqPNYXCNIw0PNK1hkunSAj
+4CA4U3A7zXkV9nF7FxpLjDWd45fnUxilXoCmcOqzALP0OLKIQ/2yPVWdicNTN4XFTmEwy2huWMMIF5bR4nQNnCAQmdtt5QkJ9WVSZ83LQcCK00yiV5yQARoFQ8lNAgMBAAE=
pubcert:
MIIGazCCBFOgAwIBAgITMwAAAA6GfbGZe5fD8AAAAAAADjANBgkqhkiG9w0BAQwFADBVMQswCQYDVQQGEwJVUzEeMBwGA1UEChMVTWljcm9zb2Z0IENvcnBvcmF0aW9uMSYwJA
YDVQQDEx1NaWNyb3NvZnQgU0NEIFByb2R1Y3RzIFJTQSBDQTAeFw0yMzAxMDUxOTIyNDdaFw0yNDAxMDMxOTIyNDdaMGwxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpXYXNoaW5n
dG9uMRAwDgYDVQQHEwdSZWRtb25kMR4wHAYDVQQKExVNaWNyb3NvZnQgQ29ycG9yYXRpb24xFjAUBgNVBAMTDUNvbnRhaW5lclBsYXQwggGiMA0GCSqGSIb3DQEBAQUAA4IBj
wAwggGKAoIBgQC1zPF0G+N8JW8J6+Ow0Fy2zmp7/L50bVxNcPwZ7Zc2Q0D3WDCTG9AHY2hAwWZdGS+kfsP/O+F9rUt7XXRh3NlXKQo3h5HCHxRl8sewhWj8mMTvPiAcLplfkc41bxjZ6
jD1nHlRZvjRIqjKP4swITqyuELLFv/3dFgFMoRHud210PCGCrQ5C2kVHCO3ROFO1RHNEwoOwB4ahp7H4qflxW5fPcKtfoAHdEEOcSDsPxAecJGNZZHmGV15kJ8yqZsGNDCBzJ8dXKi2lvz
UEI1sC1zQrU2LHkcHyW75vZfI7y8GISQD+/r8kDTSCD8jUyIX75QHkNhtcZiTY87JAct7zQTQFQOiC+WzNyvRZqhGi+LmKUkmGo81hcI0jDQ80rWGS6dICP7gIDhTcDvNeRX2cXsXGkuMNZ
3jl+dTGKVegKZw6rMAs/Q4sohD/bI9VZ2Jw1M3hcVOYTDLaG5YwwgXltHidA2cIBCZ223lCQn1ZVJnzctBwIrTTKJXnJABGgVDyU0CAwEAAaOCAZswggGXMA4GA1UdDwEB/wQEAwIHgDAj
BgNVHSUEHDAaBgsrBgEEAYI3TDsBAQYLKwYBBAGCN0w7AQIwHQYDVR0OBBYEFBZGoCKzvZk9Mx5xSX25m/u2+IArMEUGA1UdEQQ+MDykOjA4MR4wHAYDVQQLExVNaWNyb3NvZ
nQgQ29ycG9yYXRpb24xFjAUBgNVBAUTDTQ3Mjk3Mis1MDAwNDUwHwYDVR0jBBgwFoAUVc1NhW7NSjXDjj9yAbqqmBmXS6cwXgYDVR0fBFcwVTBToFGgT4ZNaHR0cDovL3d3dy5taW
Nyb3NvZnQuY29tL3BraW9wcy9jcmwvTWljcm9zb2Z0JTIwU0NEJTIwUHJvZHVjdHMlMjBSU0ElMjBDQS5jcmwwawYIKwYBBQUHAQEEXzBdMFsGCCsGAQUFBzAChk9odHRwOi8vd3d3
Lm1pY3Jvc29mdC5jb20vcGtpb3BzL2NlcnRzL01pY3Jvc29mdCUyMFNDRCUyMFByb2R1Y3RzJTIwUlNBJTIwQ0EuY3J0MAwGA1UdEwEB/wQCMAAwDQYJKoZIhvcNAQEMBQADggIBAG
UC0GBN5viiEbQWT2Mv1BNN95rYHcdtEgYbGr9Pkks46FY4wPys4YzDR6m7hB2JiMbXMsyzldgjZdjYQniW5/yv4QyH02cXSWrqR50NSq4QKpsx+jVwvlZXB3le6zbqmnNAWz+MsXxS4XKO
giV3ND1BJ0163zt5+fX94gTyno4b39+oND1jY0N20AWupTC9LoeWZcxvXi3/Nf40w5ugANHXB6WAqQSmv1EudOyB9xzoBDe0voafm0F8Y6r9gj/KL6F5Qi7ZWEfk22z1trYOw2cYDwnH3uG
NW5kev9cvzEP5WrkYZxJcj/00fzTfJ9H6iYRvvxwmQuRsuj9mLjgNVBSpnbATrdTtuZ7jIc0VQsMgtJFR8I1pbTIOZdD02J/FCiJYyox+Vqq+yuDLy+00q4dHuQOYoaRskQCOtKoaPBd0Y1RG6Dv
KxUtcotC2UTSvTWndQjxcnvPaGLr4QGJEiMw7Rnn4QK+x+8V8jBO8am0cUFr2Qa6xEhwHk+1Pf7pOnBJ6/SjyGzLTfpdGD4L7yQZ4eQFHono5+7KvmA/hFow+cnl8FPRi0UqZ01UoAuQz8h
0XMyXqytE24zJuosJv/kfpU7g3ohASr7LwgJvbzTyZmwrCe4Lh43cW9z4ADxYSCMptWrKddNA4xy0Hq+uPAzRV3BesuHYDHLAmQOHINW9x
payload:
{"x-ms-sevsnpvm-guestsvn": “100",
  "x-ms-sevsnpvm-launchmeasurement": "d527e299c6d2d1b16575656d37232f2733c0d8b3c3b923eba9f4a95eaec544d5f0398ae7c1ee8d31fca5da5f82befefa"
}
```