Rule Language

The USBGuard daemon decides which USB device to authorize based on a policy defined by a set of rules. When an USB device is inserted into the system, the daemon scans the existing rules sequentially and when a matching rule is found, it either authorizes (allows), deauthorizes (blocks) or removes (rejects) the device, based on the rule target. If no matching rule is found, the decision is based on an implicit default target. This implicit default is to block the device until a decision is made by the user.

The rule language grammar, expressed in a BNF-like syntax, is the following:

```
rule ::= target device_id device_attributes conditions.

target ::= "allow" | "block" | "reject".

device_id ::= "*:*" | vendor_id ":*" | vendor_id ":" product_id.

device_attributes ::= device_attributes | attribute.
device_attributes ::= .

conditions ::= conditions | condition.
conditions ::= .
```

See Device attributes section for the list of available attributes and Conditions for the list of supported rule conditions.

Targets

The target of a rule specifies whether the device will be authorized for use or not. Three types of target are recognized:

- `allow` - authorize the device
- `block` - deauthorize the device
- `reject` - remove the device from the system

Device specification

Except the target, all the other fields of a rule need not be specified. Such a minimal rule will match any device and allows the policy creator to write an explicit default target (there's an implicit one too, more on that later). However, if one want's to narrow the applicability of a rule to a set of devices or one device only, it's possible to do so with a device id and/or device attributes.

Device ID

A USB device ID is the colon delimited pair *vendor_id:product_id*. All USB devices have this ID assigned by the manufacturer and it should uniquely identify a USB product. Both *vendor_id* and *product_id* are 16-bit numbers represented in hexadecimal base.

In the rule, it's possible to use an asterisk character to match either any device ID `*:*` or any product ID from a specific vendor, e.g. `1234:*`.

Device attributes

(Please see issue #11 and comment on the changes related to this section)

Device attributes are specific value read from the USB device after it's inserted to the system. Which attributes are available is defined bellow. Some of the attributes are derived or based on attributes read directly from the device. The value of an attribute is represented as a double-quoted string.

List of attributes:

- `hash "[0-9a-f]{32}"`: Match a hash of the device attributes (the hash is computed for every device by USBGuard).
- `name "device-name"`: Match the USB device name attribute.
- `serial "serial-number"`: Match the iSerial USB device attribute.
- `via-port "port-id"`: Match the USB port through which the device is connected.
- `via-port [operator] { "port-id" "port-id" ... }`: Match a set of USB ports.
- `with-interface interface-type`: Match an interface the USB device provides.
- `with-interface [operator] { interface-type interface-type ... }`: Match a set of interface types against the set of interfaces that the USB device provides.

`operator` is one of:

- `all-of`: The device attribute set must contain all of the specified values for the rule to match.
- `one-of`: The device attribute set must contain at least one of the specified values for the rule to match.
- `none-of`: The device attribute set must not contain any of the specified values for the rule to match.
- `equals`: The device attribute set must contain exactly the same set of values for the rule to match.
- `equals-ordered`: The device attribute set must contain exactly the same set of values in the same order for the rule to match.

`port-id` is a platform specific USB port identification. On Linux it's in the form "b-n" where `b` and `n` are unsigned integers (e.g. "1-2", "2-4", …).

`interface-type` represents a USB interface and should be formated as three 8-bit numbers in hexadecimal base delimited by colon, i.e. `cc:ss:pp`. The numbers represent the interface class (`cc`), subclass (`ss`) and protocol (`pp`) as assigned by the USB-IF (List of assigned classes, subclasses and protocols). Instead of the subclass and protocol number, you may write an asterisk character (`\*`) to match all subclasses or protocols. Matching a specific class and a specific protocol is not allowed, i.e. if you use an asterisk as the subclass number, you have to use an asterisk for the protocol too.

Conditions

Whether a rule that matches a device will be applied or not can be further restricted using rule conditions. If the condition expression is met at the rule evaluation time, then the rule target is applied for the device. A condition expression is met if it evaluates to true. Otherwise, the rule evaluation continues with the next rule. A rule conditions has the following syntax:

```
if [!]condition
if [operator] { [!]conditionA [!]conditionB ... }
```

Optionally, an exclamation mark (`!`) can be used to negate the result of a condition.

Interpretation of the set operator:

- `all-of`: Evaluate to true if all of the specified conditions evaluated to true.
- `one-of`: Evaluate to true if one of the specified conditions evaluated to true.
- `none-of`: Evaluate to true if none of the specified conditions evaluated to true.
- `equals`: Same as `all-of`.
- `equals-ordered`: Same as `all-of`.

List of conditions:

- `localtime(time_range)`: Evaluates to true if the local time is in the specified time range. `time_range` can be written either as `HH:MM[:SS]` or `HH:MM[:SS]-HH:MM[:SS]`.
- `allowed-matches(query)`: Evaluates to true if an allowed device matches the specified query. The query uses the rule syntax. **Conditions in the query are not evaluated**.
- `rule-applied`: Evaluates to true if the rule currently being evaluated ever matched a device.
- `rule-applied(past_duration)`: Evaluates to true if the rule currently being evaluated matched a device in the past duration of time specified by the parameter. `past_duration` can be written as `HH:MM:SS`, `HH:MM`, or `SS`.
- `rule-evaluated`: Evaluates to true if the rule currently being evaluated was ever evaluated before.
- `rule-evaluated(past_duration)`: Evaluates to true if the rule currently being evaluated was evaluated in the pas duration of time specified by the parameter. `past_duration` can be written as `HH:MM:SS`, `HH:MM`, or `SS`.

- `random`: Evaluates to true/false with a probability of `p=0.5`.
- `random(p_true)`: Evaluates to true with the specified probability `p_true`.
- `true`: Evaluates always to true.
- `false`: Evaluates always to false.

Initial policy

Using the `usbguard` CLI tool and its `generate-policy` subcommand, you can generate an initial policy for your system instead of writing one from scratch. The tool generates an **allow** policy for all devices connected to the system at the moment of execution. It has several options to tweak the resulting policy:

- `-p`: Generate port specific rules for all devices. By default, port specific rules are generated only for devices which do not export an iSerial value. See the `-P` option for more details.
- `-P`: Don't generate port specific rules for devices without an iSerial value. Without this option, the tool will add a `via-port` attribute to any device that doesn't provide a serial number. This is a security measure to limit devices that cannot be uniquely identified to connect only via a specific port. This makes it harder to bypass the policy since the real device will ocupy the allowed USB port most of the time.
- `-t <target>`: Generate an explicit "catch all" rule with the specified target. The target can be one of the following values: `allow`, `block`, `reject`.
- `-X`: Don't generate a hash attribute for each device.
- `-H`: Generate a hash-only policy.

The policy will be printed out on the standard output. It's a good idea to review the generated rules before using them on a system. The typical workflow for generating an initial policy could look like this:

```
# usbguard generate-policy > rules.conf
# vi rules.conf
(review/modify the rule set)
# sudo install -m 0600 -o root -g root rules.conf /etc/usbguard/rules.conf
# sudo systemctl restart usbguard
```

Example policies

The following examples show what to put into the `rules.conf` file in order to implement the given policy.

Allow USB mass storage devices (USB flash disks) and block everything else

This policy will block any device that isn't just a mass storage device. Devices with a hidden keyboard interface in a USB flash disk will be blocked. Only devices with a single mass storage interface will be allowed to interact with the operating system. The policy consists of a single rule:

```
allow with-interface equals { 08:*:* }
```

The blocking is implicit in this case because we didn't write a `block` rule. Implicit blocking is useful to desktop users because a desktop applet listening to USBGuard events can ask the user for a decision if an implicit target was selected for a device.

Allow a specific Yubikey device to be connected via a specific port. Reject everything else on that port.

```
allow 1050:0011 name "Yubico Yubikey II" serial "0001234567" via-port "1-2" hash "044b5e168d40ee0245478416caf3d998"
reject via-port "1-2"
```

We could use just the hash to match the device. However, using the name and serial attributes allows the policy creator to quickly assign rules to specific devices without computing the hash. On the other hand, the hash is the most specific value we can use to identify a device in USBGuard so it's the best attribute to use if you want a rule to match just one device.

Reject devices with suspicious combination of interfaces

A USB flash disk which implements a keyboard or a network interface is very suspicious. The following set of rules forms a policy which allows USB flash disks and explicitly rejects devices with an additional and suspicious (as defined before) interface.

```
allow with-interface equals { 08:*:* }
reject with-interface all-of { 08:*:* 03:00:* }
reject with-interface all-of { 08:*:* 03:01:* }
```

```
reject with-interface all-of { 08:*:* e0:*:* }
reject with-interface all-of { 08:*:* 02:*:* }
```

The policy rejects all USB flash disk devices with an interface from the HID/Keyboard, Communications and Wireless classes. Please note that blacklisting is the wrong approach and you shouldn't just blacklist a set of devices and allow the rest. The policy above assumes that blocking is the implicit default. Rejecting a set of devices considered as "bad" is a good approach how to limit the exposure of the OS to such devices as much as possible.

Allow a keyboard-only USB device only if there isn't already a USB device with a keyboard interface allowed

```
allow with-interface one-of { 03:00:01 03:01:01 } if !allowed-matches(with-interface one-of { 03:00:01 03:01:01 })
```

Play "Russian roulette" with USB devices

```
allow if random(0.1666)
reject
```