YML Graph Definition Spec

22-03-17

Table of Contents

This document describes how to define a model of a system in YAML.

1. Layers

Layers are just defined by their name. A *type* field is required to distinguish the yaml definition from other types of specs. It's also necessary to specify an *upper* and *lower* layer name to create the layer hierarchy. If *upper* or *lower* definitions are omitted, it's assumed as *None*

```
layername:
  type: layer
  upper: layer_above|None
  lower: layer_below|None
```

2. Links

The simplest form of link definition is within a node spec. In this context it's only necessary to specify the destination *d* because the source is already uniquely identified as the node that contains the links.

```
nodename:
 layer: layername
 links:  # from nodename to destnode with default probability
   - {d: destlayer.destnode}
```

In addition to the destination, all link definitions take a probability parameter *p* (link probabilities are currently not considered. This is an optional feature for potential future use).

```
nodename:
 layer: layername
 links:  # from nodename to destnode with probability
   - {d: destlayer.destnode, p: [0.0-1.0]}
```

With the introduction of stencil instantiation, link definitions require another level of complexity because referencing a stencil is insufficient to clearly identify the source or destination subnode of a link. Within the stencil context, links require a source *s* identifying the subnode to start from.

Destinations *d* into stencils require a 3-component address. Therefore the most complete definition of a link is as follows:

```
links:  # from a stencil instance subnode to another
 - {s: subnode_name, d: destlayer.stencil_instance.destnode, p: [0.0-1.0]}
```

3. Groups

Each node can be part of several groups. This feature allows to overwrite the specific probabilities by a group-probability and therefore enables a broad range of statistics. In addition to the predefined groups *vulnerable* and *protected*, it is possible to create user-defined groups. All names are converted to lower-case internally, so that any capitalization is ignored.

The order of the listed groups defines the priority in case of conflicting probability configurations. For example, if a node is member of groups *protected* and *userdef* and the configuration specifies different probabilities for the two groups, then the probability of the group that's listed first will take precedence.

Any nodes with dedicated probability entries will not be affected by the group probability unless a command line flag is used to make groups override the node-specific values.

Group probabilities can be specified via command line or configuration in yaml as shown below (note that it needs to be a list of dictionary entries):

```
groups:
 - vulnerable: 0.8
 - protected: 0.3
 - userdef: 0.647
```

4. Nodes

The node definition opens with the *nodename*. The minimum required fields are *layer* and *links*. Additional fields are *probability* and *groups*. Note that the group priorities are exclusively defined in the global group definition, i.e. the order within a node definition has no meaning.

```
nodename:
 layer: layername
 links:
   # regular node destination with probability
   - {d: destlayer1.destnode, p: 0.6}
   # stencil instance destination
   - {d: destlayer2.stencilinstance.destsubnode}
 groups:
   - userdef
 probability: 0.4
 impact: 2.0
```

5. Stencils

Stencils are defined by their stencil name, a *type* field, and contain a list of *nodes*. Each node defines a list of *links* (within this stencil only) and an optional node *probability*. If no node probability is provided, the stencil can define a default *probability*.

```
stencilname:
 type: stencil
 nodes:
   - node1:
```

```
      links:
        - node2
        - node3
      probability: [0.0-1.0]
  - node2:
      links:
        - node1
      probability: [0.0-1.0]
  - node3:
      links:
        - node2
        - node1
      probability: [0.0-1.0]
probability: [0.0-1.0]   (default for nodes)
```

6. Stencil Instantiation

Stencil instantiations are the actual node definitions that get included into the graph based on a defined stencil. Instantiations share most of their details with nodes (e.g. *layer*, *links*, *groups*, or *probability* specifications).

In addition to nodes, they require a reference to the *stencil* template name and an optional *modifier*. Modifiers allow to change the names and probabilities of stencil subnodes so that a stencil template can be adjusted to a given situation without requiring to define a new stencil template. It also allows to replicate subnodes. For example, our **process** stencil has a connector subnode for communication. If a process instance requires more communication subnodes for multiple specific protocols, the communication node can be replicated using a modifier.

The example below defines a stencil instance of the process template and creates a node for *grpc* and *uds* with given probabilities in the place of the original *comm* subnode. The intra-stencil connectivity will be replicated for each of the newly defined subnodes as well. The example also shows that the *grpc* subnode is added to a user-defined node group $grpc_{group}$ to enable the flexibility of setting probabilities for groups (e.g. to avoid editing all occurrences of grpc in this and other stencils and nodes)

The example also shows how to remove the subnode (*files*) from a stencil template or rename subnode *memory* to *ram* for this particular instance. This might be useful for scenarios, where certain predefined subnodes are disabled or disconnected from a templated component or should get a dedicated name and group for better overview - again without the need to make a new stencil for just this one instance.

Parts of stencil instances can also be *collapsed*. The advantage of collapsing is to reduce the amount of vertices and paths while keeping the consistency and convenience of stencil nodes that serve as connectors. Only one collapsed node can be created from a list of directly connected nodes. Otherwise, if the sub-graph of collapsed nodes has disconnected nodes, internal and external links cannot be correctly connected. Inbound and outbound links to nodes that are collapsed will be remapped to the new grouped node.

The field *cprobability* allow to assign a specific probability to the collapsed node. Otherwise, the weighted score of collapsed nodes will be used where the weights reflect the number of links in and out of the sub-node.

Consideration: If stencil is (partially) collapsed, then the 'edge'-node probabilities become the external link probabilities (once we add this feature to the tool).

```
myprocess:
 stencil: process
```

```
layer: thislayer
modifier:
  - comm:
      - {n: grpc, p: 0.4, g:grpc_group}
      - {n: uds, p: 0.45}
  - files: []
  - memory:
      - {n: ram}
links:
  - {s: grpc, d: thislayer.theother.grpc}
  - {s: uds, d: otherlayer.logserver.comm}
collapsed:
  - core
  - syscall
cprobability: 0.5
cimpact: 3.5
probability: 0.62
collapsed: yes
```

7. Other Configuration

7.1. Attacker and Target

Two sections in your top-level file may define which nodes should be used as the *target* of attacks and which should be used as *attacker* nodes. Both require the full node address including layer, node name, and stencil instance name (in case it is a subnode of a stencil). Alternatively, those node types can also be defined via command line argument.

```
attacker:
 - layerA.stencilinstance.nodenameA
 - layerB.nodenameB

target:
 - layerC.nodenameC
```

7.2. Include files

It's possible to split your graph definitions into multiple files. For example to keep stencil definitions, layers, and configuration separate for better reuse. Your top-level file can then use an *include* section to combine the files into one model.

The example below separates a 3-layer model into one file for each layer (*services.yml*, *os.yml*, *hw.yml*) plus a stencil template file *stencils.yml* and the hierarchy definition *layers.yml*.

```
includes:
 - stencils.yml
 - layers.yml
 - service.yml
 - os.yml
 - hw.yml
```

Another idea to split the files could be to make the hierarchy definition your top-level file and include the stencils and layers.

```
includes:
 - stencils.yml
 - service.yml
 - os.yml
```

```
 - hw.yml

services:
 type: layer
 upper: None
 lower: os

os:
 type: layer
 upper: services
 lower: hw

hw:
 type: layer
 upper: os
 lower: hw
```

Misusing YAML in this way imposes a limitation that there can be only one such section within the entire list of involved files. Otherwise, a later include section would overwrite any previous one, i.e. the include definitions do not 'accumulate'. That said, the actual implementation will only handle includes that appear in the top level input file.