SkilletLib

Release 0.1

May 11, 2023

Contents:

CHAPTER 1

Examples

1.1 Validation Examples

Validation Skillets work by comparing the 'configuration objects' against a set of rules defined in each snippet. By default, the configuration is placed in the context as the 'config' variable and is a string representation of the raw XML. To compare and validate specific parts of the config, you can use the 'parse' command to convert a part of the configuration into an object that can then be used with simple logical operators or jinja filters.

1.1.1 Variable Parsing

This example captures a variable called *update_schedule_object* by converting the configuration elements found at the given *xpath* from the *config* variable. The output of this snippet is a new variable is placed into the context and is available for use in subsequent steps.

```
- name: create_update_schedule_object
  cmd: parse
  variable: config
  outputs:
    - name: update_schedule_object
      capture_object: /config/devices/entry[@name='localhost.localdomain']/
↪deviceconfig/system/update-schedule
```

The *updated_schedule_object* variable will contain all the configuration elements found at that xpath:

```
<update-schedule>
 <statistics-service>
   <application-reports>yes</application-reports>
   <threat-prevention-reports>yes</threat-prevention-reports>
   <threat-prevention-pcap>yes</threat-prevention-pcap>
   <threat-prevention-information>yes</threat-prevention-information>
   <passive-dns-monitoring>yes</passive-dns-monitoring>
   <url-reports>yes</url-reports>
```

(continues on next page)

(continued from previous page)

```
    <health-performance-reports>yes</health-performance-reports>
    <file-identification-reports>yes</file-identification-reports>
  </statistics-service>
  <threats>
    <recurring>
      <every-30-mins>
        <at>2</at>
        <action>download-and-install</action>
      </every-30-mins>
      <threshold>48</threshold>
    </recurring>
  </threats>
  <anti-virus>
    <recurring>
      <hourly>
        <at>4</at>
        <action>download-and-install</action>
      </hourly>
    </recurring>
  </anti-virus>
  <wildfire>
    <recurring>
      <every-min>
        <action>download-and-install</action>
      </every-min>
    </recurring>
  </wildfire>
  <global-protect-datafile>
    <recurring>
      <hourly>
        <at>40</at>
        <action>download-and-install</action>
      </hourly>
    </recurring>
  </global-protect-datafile>
  <global-protect-clientless-vpn>
    <recurring>
      <hourly>
        <at>50</at>
        <action>download-and-install</action>
      </hourly>
    </recurring>
  </global-protect-clientless-vpn>
</update-schedule>
```

The previous XML fragment will be converted into an object with name *update_schedule_object* with the following value:

```
{
  "update-schedule": {
    "threats": {
      "recurring": {
        "every-30-mins": {
          "at": "2",
          "action": "download-and-install"
        },
```

(continues on next page)

(continued from previous page)

```
        "threshold": "48"
      }
    },
    "statistics-service": {
      "application-reports": "yes",
      "threat-prevention-reports": "yes",
      "threat-prevention-pcap": "yes",
      "threat-prevention-information": "yes",
      "passive-dns-monitoring": "yes",
      "url-reports": "yes",
      "health-performance-reports": "yes",
      "file-identification-reports": "yes"
    },
    "anti-virus": {
      "recurring": {
        "hourly": {
          "at": "4",
          "action": "download-and-install"
        }
      }
    },
    "wildfire": {
      "recurring": {
        "every-min": {
          "action": "download-and-install"
        }
      }
    },
    "global-protect-datafile": {
      "recurring": {
        "hourly": {
          "at": "40",
          "action": "download-and-install"
        }
      }
    },
    "global-protect-clientless-vpn": {
      "recurring": {
        "hourly": {
          "at": "50",
          "action": "download-and-install"
        }
      }
    }
  }
}
```

1.1.2 Validation

The *validation* cmd type can be used to validate configuration objects with simple logical operators and Jinja filters. This example will validate that a configuration node is present on the *update_schedule_object* variable.

```
- name: update_schedule_stats_service_configured
  when: update_schedule_object is not none
  label: Ensure Statistics Service is enabled
```

(continues on next page)

(continued from previous page)

```
  test: update_schedule_object| node_present('update-schedule.statistics-service')
  documentation_link: https://docs.paloaltonetworks.com/pan-os/8-0/pan-os-new-
↪features/content-inspection-features/telemetry-and-threat-intelligence-sharing
```

Note: See the *Jinja Filters* for details on available filters.

1.1.3 XML Validate

The *validate_xml* cmd type can be used to compare the configuration against an XML Snippet either in whole, or against a smaller portion of the XML Fragment using *cherry_pick*.

This will query the configuration for the XML Element at the given XPath and compare it against the contents of the 'file' or 'element' attributes. The *file* attribute, if found, will be rendered using Jinja and stored in the *element* attribute for comparison. The *file* or *element* must be rooted at the same xpath. If you have many validations to perform in the same area of the configuration, you can use *cherry_pick* to validate portions of a larger XML *file* or *element*.

```
# this example will validate that the application-reports xml fragment matches that␣
↪that is found in the
# device_system.xml file
- name: validate_application_reports
  cmd: validate_xml
  xpath: /config/devices/entry[@name='localhost.localdomain']/deviceconfig/system
  file: device_system.xml
  cherry_pick: update-schedule/statistics-service/application-reports

- name: validate_statistics_service
  cmd: validate_xml
  xpath: /config/devices/entry[@name='localhost.localdomain']/deviceconfig/system
  file: device_system.xml
  cherry_pick: update-schedule/statistics-service

- name: validate_update_anti_virus
  cmd: validate_xml
  xpath: /config/devices/entry[@name='localhost.localdomain']/deviceconfig/system/
↪update-schedule/anti-virus
  file: anti_virus.xml
```

CHAPTER 2

Skilletlib Skillet Types

2.1 Panos Skillet

class skilletlib.skillet.panos.**PanosSkillet**(*metadata: dict*, *panoply: skilletlib.panoply.Panoply = None*)

get_results() → dict

PanosSkillet will return a dict containing three keys: result, changed, and snippets. If any snippet failed, the result will be 'failure' otherwise 'success' If any successful snippet may have caused a change to the device, the 'changed' attribute will be 'True'

A skillet that contains only the following snippet, will generate the output below:

```
- name: check_hostname_again
  cmd: op
  cmd_str: <show><system><info/></system></show>
  outputs:
    - name: url-db
      capture_pattern: ./url-db
    - name: pa-version
      capture_pattern: ./plugin_versions/entry[@name="cloud_services"]/
↪@version
```

```
{
  'snippets': {
    'check_hostname_again': {
      'results': 'success',
      'changed': True
    }
  },
  'outputs': {
    'url-db': 'paloaltonetworks',
    'pa-version': '1.5.0'
  },
```

(continues on next page)

(continued from previous page)

```
  'result': 'success',
  'changed': True
}
```

Returns dict containing default outputs plus the overall result and changed flag

get_snippets() → List[skilletlib.snippet.panos.PanosSnippet]
Perform Panos Skillet specific tasks while loading each snippet

Returns a List of PanosSnippets

initialize_context(*initial_context: dict*) → dict
In this panos case, we want to stash the current configuration of the panos device in question in the context, check for online mode, offline mode, or an existing panoply object

Parameters **initial_context** – dict to use to initialize the context

Returns context with additional initialized items

static load_element(*snippet_def: dict*, *snippet_path: pathlib.Path*) → dict
This method will load the snippet file found on disk into the 'element' attribute if the element is not already populated. This allows snippets to be 'all-in-one' i.e. there is no requirement for the snippets to be split into separate files. The meta-cnc.yaml file can contain all the snippets 'inline' in the 'element' attribute if desired. An example snippet def:

- name: template xpath: /config/devices/entry[@name='localhost.localdomain']/template file: ../snippets/template.xml

Parameters **snippet_def** – the loaded snippet definition from the skillet.yaml file. Each snippet object in the

'snippets' stanza is a snippet_def and is passed in here :param snippet_path: the path on the filesystem where this skillet is located. This is used to resolve relative paths for each snippet. This allows snippet file re-use across skillets. :return: snippet_def with the element populated with the resolved and loaded snippet file contents

2.2 PAN Validation Skillet

class skilletlib.skillet.pan_validation.**PanValidationSkillet**(*metadata: dict*, *panoply: skilletlib.panoply.Panoply = None*)

get_results() → dict
Pan-validation skillets return a dictionary with a key for each test that was executed. Each value of those keys will be a dict containing the following keys:

- results - whether the test conditional was true or false
- label - human readable label of the test
- severity - a string that may be set to indicate the severity of a test
- documentation_link - an HTTP link where the user can get more information about this test
- output_message - A rendered output message regarding the test results

```
{
    "update_schedule_configured": {
        "results": true,
        "label": "Ensure Update Schedules are Configured",
        "severity": "low",
        "documentation_link": "https://iron-skillet.readthedocs.io",
        "test": "update_schedule_object is not none",
        "output_message": "Snippet Validation Passed"
    },
}
```

Returns dictionary with the aforementioned keys

get_snippets() → List[skilletlib.snippet.pan_validation.PanValidationSnippet]
Perform Panos Skillet specific tasks while loading each snippet

Returns a List of PanosSnippets

2.3 REST Skillet

class skilletlib.skillet.rest.**RestSkillet**(*metadata: dict*)

get_results() → dict
Gets the results from the REST skillet execution. This skillet does not add any additional attributes to the normal output.

The following snippet will generate the following output:

```
- name: Retrieve Remote Network Service IP from Prisma Access
  path: https://api.gpcloudservice.com/getAddrList/latest?fwType=gpcs_remote_
↪network&addrType=public_ip
  operation: GET
  headers:
    header-api-key: '{{ api_key }}'
  output_type: json
  outputs:
    - name: status
      capture_pattern: $.status
    - name: fwType
      capture_pattern: $.result.fwType
    - name: addrList
      capture_pattern: $.result.addrList
```

```
{
    'snippets': {
        'Retrieve Remote Network Service IP from Prisma Access': {
            'results': 'success',
            'raw': {
                'status': 'success',
                'result': {
                    'fwType': 'gpcs_remote_network',
                    'addrListType': 'public_ip',
                    'total-count': 2,
                    'addrList': [
```

(continues on next page)

(continued from previous page)

```
                        'test-XXX:x.x.x.x',
                        'pa220-test, pa220-test-2:x.x.x.x'
                    ]
                }
            }
        }
    },
    'outputs': {
        'status': 'success',
        'fwType': 'gpcs_remote_network',
        'addrList': "['test-XXX:x.x.x.x', 'pa220-test, pa220-test-2:x.x.x.x']"
    }
}
```

Returns dictionary of results from the REST Skillet execute or execute_async method

get_snippets() → List[skilletlib.snippet.base.Snippet]
Loads and validates each Snippet in the REST skillet metadata file

Returns List of Snippets for this Skillet Class

2.4 Template Skillet

class skilletlib.skillet.template.**TemplateSkillet**(*s: dict*)

get_results() → dict
TemplateSkillet will add an additional attribute into the results dict containing the value of the first snippet found to have been successfully executed

```
{
  "snippets": {
    "config_template": "success"
  },
  "template": "Rendered Template output"
}
```

Returns dict containing default outputs plus the rendered template contained in the 'template' attribute

get_snippets() → List[skilletlib.snippet.template.TemplateSnippet]
Each skillet determines how it's snippets are to be loaded and initialized. Each Skillet type must implement this method.

Returns List of Snippets for this Skillet Class

2.5 Terraform Skillet

class skilletlib.skillet.terraform.**TerraformSkillet**(*s: dict*)

get_snippets() → List[skilletlib.snippet.base.Snippet]
Each skillet determines how it's snippets are to be loaded and initialized. Each Skillet type must implement this method.

Returns List of Snippets for this Skillet Class

2.6 Base Skillet

This is base class from which all Skillets derive.

class skilletlib.skillet.base.**Skillet**(*s: dict*)

dump_yaml() → str
Convert this Skillet into a YAML formatted string

Returns YAML formatted string

execute(*initial_context: dict*) → dict
The heart of the Skillet class. This method executes the skillet by iterating over all the skillets returned from the 'get_skillets' method. Each one is checked if it should be executed if a 'when' conditional attribute is found, and if so, is executed using the snippet execute method.

Parameters **initial_context** – context of key values pairs to use for the execution. By default this is all the

variables defined in the skillet file with their default values. Updates from user input, the environment, etc will override these default values via the 'update_context' method. :return: a dict containing the updated context containing the output of each of the snippets

execute_async(*initial_context: dict*) → Generator
Returns a generator that can be used to iterate over the output as it's generated from each snippet. The calling application should call 'get_results' once the execute is complete

Parameters **initial_context** – context of key values pairs to use for the execution. By default this is all the

variables defined in the skillet file with their default values. Updates from user input, the environment, etc will override these default values via the 'update_context' method. :return: generator[str]

get_declared_variables() → List[str]
Return a list of all variables defined in all the snippets that are not defined as an output

Returns list of variable names

get_results() → dict
Returns the results from the skillet execution. This must be called manually if using 'execute_async'. The returned dict will include a 'snippets' dictionary that contains a key for each snippet that was executed. Each snippet dictionary will contain the 'results' and 'raw' attributes.

```
{
  'snippets': {
    'check_hostname_again': {
      'results': 'success',
      'changed': True
    }
  },
  'outputs': {
    'url-db': 'paloaltonetworks',
```

(continues on next page)

(continued from previous page)

```
    'pa-version': '1.5.0'
  },
  'result': 'success',
  'changed': True
}
```

Returns dictionary of results from the Skillet execute or execute_async method

get_snippet_by_name(*snippet_name: str*) → skilletlib.snippet.base.Snippet
Utility method to return the snippet with snippet_name

Parameters **snippet_name** – name attribute of the snippet to return

Returns Snippet Object

get_snippets() → List[skilletlib.snippet.base.Snippet]
Each skillet determines how it's snippets are to be loaded and initialized. Each Skillet type must implement this method.

Returns List of Snippets for this Skillet Class

get_variable_by_name(*variable_name: str*) → dict
Utility method to return the variable with tne variable_name

Parameters **variable_name** – name attribute of the variable to return

Returns dictionary of variable options

initialize_context(*initial_context: dict*) → dict
Child classes can override this to provide any initialization information in the context. For example, 'panos' skillets use this to set up and initialize a Panos device object

Parameters **initial_context** – Initial Context from user input, environment vars, etc

Returns updated context with initial context items plus any initialization items

load_template(*template_path: str*) → str
Utility method to load a template file and return the contents as str

Parameters **template_path** – relative path to the template to load

Returns str contents

update_context(*d: dict*) → dict
Take the input dict d and update the skillet context. I.e. any variables passed in via environment variables will be used to update the context stored on this skillet.

Parameters **d** – dictionary of key value pairs. Any keys that match 'variable' keys will be used to update the context

Returns updated context stored on this skillet

CHAPTER 3

Snippetlib Snippet Types

3.1 Panos Snippet

class skilletlib.snippet.panos.**PanosSnippet**(*metadata: dict*, *panoply: skilletlib.panoply.Panoply*)

add_filters() → None
Each snippet sub-class can add additional filters. See the PanosSnippet for examples

Returns None

cherry_pick_element(*element: str*, *cherry_pick_path: str*) → str
Cherry picking allows the skillet builder to pull out specific bits of a larger configuration and load only the smaller chunks. This is especially useful when combined with 'when' conditionals

Parameters

- **element** – string containing the jinja templated xml fragment
- **cherry_pick_path** – string describing the relative xpath to use to cherry pick an xml node from the element given as a parameter

Returns rendered and cherry_picked element

cherry_pick_xpath(*base_xpath: str*, *cherry_picked_xpath: str*) → str
When cherry picking is active, we are only going to push a smaller portion of the xml fragment. As such, we need to combine the base xpath for the xml file and the xpath to the cherry_picked node.

Ensure we can combine the base xpath and the cherry_picked xpath cleanly ideally, we end up with something like base_xpath: /config/devices/entry[@name='localhost.localdomain']/deviceconfig/system cherry_picked: update-schedule/statistics-service/application-reports Because the cherry_pick xpath will return the named element, we need to strip the last node from the xpath in this case, in order to push the cherry_picked element back to the device, we need to set the xpath to /config/devices/entry[@name='localhost.localdomain']/deviceconfig/system/update-schedule/statistics-service

Parameters

- **base_xpath** – base xpath for the xml fragment
- **cherry_picked_xpath** – relative xpath for cherry picking a portion of the xml fragment

Returns combined and rendered xpath

static compare_element_at_xpath(*config: str*, *element: str*, *xpath: str*, *context: dict*) → bool
Grab an xml fragment from the config given at xpath and compare it to this element

Parameters

- **config** – XML document string from which to pull the XML element to compare
- **element** – element to check against
- **xpath** – xpath to grab an xml fragment from the config for comparison
- **context** – jinja context used to interpolate any variables that may be present in the template

Returns bool true if they match

execute(*context: dict*) → Tuple[str, str]
Execute this Snippet and return a tuple consisting on raw output and a string representing success, failure, or running.

Each snippet sub class must override this method!

Parameters **context** – context to use for variable interpolation

Returns Tuple containing raw snippet output and string indicated success or failure

get_default_output(*results: str*, *status: str*) → dict
Override the default snippet get_default_output to not include raw results

Parameters

- **results** – raw output from snippet execution
- **status** – status of the snippet.execute method

Returns dict of default outputs

render_metadata(*context: dict*) → dict
Renders each item in the metadata using the provided context. Currently renders the xpath and element for PANOS type skillets

Parameters **context** – dict containing key value pairs to

Returns dict containing the snippet definition metadata with the attribute values rendered accordingly

sanitize_metadata(*metadata: dict*) → dict
Ensure all required keys are present in the snippet definition

Parameters **metadata** – dict

Returns dict

3.2 PAN Validation Snippet

class skilletlib.snippet.pan_validation.**PanValidationSnippet**(*metadata: dict*, *panoply: skilletlib.panoply.Panoply*)

Pan validation Snippet

execute(*context: dict*) → Tuple[str, str]

Execute method in pan_validation snippet overrides the execute method in panos to add ensure any exception caught always results in a failed test

Parameters **context** – snippet context used for tests

Returns tuple consisting of results, (success | failure)

handle_output_type_validation(*results: str*) → dict

Handle output type validation results

Parameters **results** – results from the test execution

Returns dict containing validation messages

3.3 REST Snippet

class skilletlib.snippet.rest.**RestSnippet**(*payload_str: str*, *metadata: dict*, *session: requests.sessions.Session*)

Rest Snippet

execute(*raw_context: dict*) → Tuple[str, str]

Execute this Snippet and return a tuple consisting on raw output and a string representing success, failure, or running.

Each snippet sub class must override this method!

Parameters **context** – context to use for variable interpolation

Returns Tuple containing raw snippet output and string indicated success or failure

sanitize_metadata(*metadata: dict*) → dict

Clean and sanitize metadata elements in this snippet definition

Parameters **metadata** – dict

Returns dict

3.4 Template Snippet

class skilletlib.snippet.template.**TemplateSnippet**(*template_str*, *metadata*)

TemplateSnippet implements a basic template object snippet

execute(*context: dict*) → Tuple[str, str]

Execute this Snippet and return a tuple consisting on raw output and a string representing success, failure, or running.

Each snippet sub class must override this method!

Parameters **context** – context to use for variable interpolation

Returns Tuple containing raw snippet output and string indicated success or failure

3.5 Base Snippet

This is base class from which all Snippets derive.

class skilletlib.snippet.base.**Snippet**(*metadata: dict*)

BaseSnippet implements a basic Noop snippet

add_filters() → None

Each snippet sub-class can add additional filters. See the PanosSnippet for examples

Returns None

capture_outputs(*results: str*, *status: str*) → dict

All snippet output or portions of snippet output can be captured and saved on the context as a new variable

Parameters

- **results** – the raw output from the snippet execution
- **status** – status of the snippet.execute method

Returns a dictionary containing all captured variables

execute(*context: dict*) → Tuple[str, str]

Execute this Snippet and return a tuple consisting on raw output and a string representing success, failure, or running.

Each snippet sub class must override this method!

Parameters **context** – context to use for variable interpolation

Returns Tuple containing raw snippet output and string indicated success or failure

execute_conditional(*test: str*, *context: dict*) → bool

Evaluate 'test' conditionals and return a bool

Parameters

- **test** – string of the conditional to execute
- **context** – jinja context containing previous outputs and user supplied variables

Returns boolean

get_default_output(*results: str*, *status: str*) → dict

each snippet type can override this method to provide it's own default output. This is used when there are no variables defined to be captured

Parameters

- **results** – raw output from snippet execution
- **status** – status of the snippet.execute method

Returns dict of default outputs

get_loop_parameter() → list

Returns the loop parameter for this snippet. If a loop parameter is not defined in the snippet def, this returns a list with a single blank str. Otherwise, return the value of the loop parameter as a list.

Returns value of loop_parameter from the context or a list with a single blank str

get_output() → Tuple[str, str]

get_output can be used when a snippet executes async and cannot or will not return output right away snippets that operate async must override this method

Returns Tuple containing the skillet output as a str and a str indicating success of failure

get_output_variables() → list
Returns a list of all output variables. This is used to determine if a snippet variable should be considered undeclared.

Returns list of str representing output variable names

get_snippet_variables() → list
Returns a list of variables defined in this snippet that are NOT defined as outputs

Returns list of str representing variables found in the jinja templates

get_variables_from_template(*template_str: str*) → list
Returns a list of jinja2 variable found in the template

Parameters **template_str** – jinja2 template

Returns list of variables declared in the template

is_filtered(*context*) → bool
Determines if a snippet should be available for execution based on the presence of the *__filter_snippets* object in the context. Snippets can be filtered by the following:

include_by_name: list of names to check. Only snippet names included in this list will be executed include_by_tag: list of tags to check. Only snippets with those tags will be executed include_by_regex: regular expression match. Only snippets whose name matches the regex will be executed

any snippet that does not match any of the above rules will be filtered out. The rules are inclusive OR

Parameters **context** – Snippet Context

Returns bool

render(*template_str: str*, *context: (<class 'dict'>, None)*) → str
Convenience method to quickly render a template_str using the provided context

Parameters

- **template_str** – jinja2 template to render
- **context** – context to pass to the jinja2 environment

Returns rendered string

render_metadata(*context: dict*) → dict
Each snippet sub class can override this method to perform jinja variable interpolation on various items in it's snippet definition. For example, the PanosSnippet will check the 'xpath' attribute and perform the required interpolation.

This handles regular strings, lists, and dictionaries such as:

in snippet class template_metadata = {'render_me', 'render_all', 'render_list'}

in metadata snippets:

- name: render_me_snippet render_me: render_{{ this }}
- name: render_all_snippet render_all:

  a_key: some_{{ value }} another_key: some_other_{{ value }}
- name: render_list_snippet render_list:
  - here_is_a_{{ value }}
  - another_{{ value }}

Parameters **context** – context from environment

Returns metadata with jinja rendered variables

reset_metadata()
Reset the metadata to the original metadata. This is used during looping so we can render items in the metadata on each iteration.

Returns None

sanitize_metadata(*metadata: dict*) → dict
method to sanitize metadata. Each snippet type can override this provide extra logic over and above just checking the required and optional fields

Parameters **metadata** – snippet metadata

Returns sanitized snippet metadata

should_execute(*context: dict*) → bool
Evaluate 'when' conditionals and return a bool if this snippet should be executed

Parameters **context** – jinja context containing previous outputs and user supplied variables

Returns boolean

update_context(*context: dict*) → dict
This will update the snippet context with the passed in dict. This gets called inside of 'should_execute'

Parameters **context** – dict of the outer context

Returns newly updated context

CHAPTER 4

SkilletLoader

class skilletlib.**SkilletLoader**(*path=None*)

SkilletLoader is used to find and load Skillets from their metadata files, either from the local filesystem or from a git repository URL

Parameters **path** – local relative path to search for all Skillet meta-data files

compile_skillet_dict(*skillet: dict*) → dict

Compile the skillet dictionary including any included snippets from other skillets. Included snippets and variables will be inserted into the skillet dictionary and any replacements / updates to those snippets / variables will be made before hand.

Parameters **skillet** – skillet definition dictionary

Returns full compiled skillet definition dictionary

create_skillet(*skillet_dict: dict*) → skilletlib.skillet.base.Skillet

Creates a Skillet object from the given skillet definition

Parameters **skillet_dict** – Dictionary loaded from the skillet.yaml definition file

Returns Skillet Object

static debug_skillet_structure(*skillet: dict*) → list

Verifies the structure of a skillet and returns a list of errors or warning if found

Parameters **skillet** – Skillet Definition Dictionary

Returns list of errors or warnings if found

get_skillet_with_name(*skillet_name: str, include_resolved_skillets=False) -> (<class 'skilletlib.skillet.base.Skillet'>, None*)

Returns a single skillet from the loaded skillets list that has the matching 'name' attribute

Parameters

- **skillet_name** – Name of the skillet to return
- **include_resolved_skillets** – boolean of whether to also check the resolved skillet list

Returns Skillet or None

load_all_label_values(*label_name: str*) → list
Returns a list of label values defined across all snippets with a given label for example:

labels: label_name: label_value

will add 'label_value' to the list

Parameters **label_name** – name of the label to search for

Returns list of strings representing all found label values for given key

load_all_skillets_from_dir(*directory: (<class 'str'>, <class 'pathlib.Path'>)*) → List[skilletlib.skillet.base.Skillet]
Recursively iterate through all sub-directories and locate all found skillets Returns a list of Loaded Skillets

Parameters **directory** – parent directory in which to start iterating

Returns list of Skillet objects

load_from_git(*repo_url*, *repo_name*, *repo_branch*, *local_dir=None*) → List[skilletlib.skillet.base.Skillet]
Performs a local clone of the given Git repository URL and returns a list of all found skillets defined therein.

Parameters

- **repo_url** – Repository URL
- **repo_name** – name given to the repository
- **repo_branch** – branch to checkout
- **local_dir** – local directory where to clone the git repository into

Returns List of Skillets

load_skillet(*skillet_path: str*) → skilletlib.skillet.base.Skillet
Returns a Skillet object from the given path

Parameters **skillet_path** – full path to the skillet YAML file

Returns Skillet object

load_skillet_dict_from_path(*skillet_path: (<class 'str'>, <class 'pathlib.Path'>)*) → dict
Loads the skillet metadata file into a skillet_dict dictionary

Parameters **skillet_path** – path in which to look for a metadata file

Returns skillet dictionary

load_skillet_dicts_from_git(*repo_url*, *repo_name*, *repo_branch*, *local_dir=None*) → List[dict]
Performs a local clone of the given Git repository URL and returns a list of all found skillet definition dictionaries defined therein.

Parameters

- **repo_url** – Repository URL
- **repo_name** – name given to the repository
- **repo_branch** – branch to checkout
- **local_dir** – local directory where to clone the git repository into

Returns List of Skillets

load_skillet_from_path(*skillet_path: (<class 'str'>, <class 'pathlib.Path'>)*) → skilletlib.skillet.base.Skillet

Returns a Skillet object from the given path.

Parameters **skillet_path** – path in which to search for a skillet

Returns Skillet object of the correct type

load_skillets_from_git(*repo_url*, *repo_name*, *repo_branch*, *local_dir=None*) → List[skilletlib.skillet.base.Skillet]

Performs a local clone of the given Git repository URL and returns a list of all found skillets defined therein.

Parameters

- **repo_url** – Repository URL
- **repo_name** – name given to the repository
- **repo_branch** – branch to checkout
- **local_dir** – local directory where to clone the git repository into

Returns List of Skillets

static normalize_skillet_dict(*skillet: dict*) → dict

Attempt to resolve common configuration file format errors

Parameters **skillet** – a loaded skillet/snippet

Returns skillet/snippet that has been 'fixed'

resolved_skillets

alias of `typing.List`

skillets

alias of `typing.List`

CHAPTER 5

Panoply

class skilletlib.panoply.**EphemeralPanos**(*hostname: Optional[str] = None*, *api_username: Optional[str] = None*, *api_password: Optional[str] = None*, *api_port: Optional[int] = 443*, *serial_number: Optional[str] = None*, *debug: Optional[bool] = False*, *api_key: Optional[str] = None*)

EphemeralPanos is used when a Panos instance may or may not be online and available at all times. This is useful when instantiating instances via a CI/CD pipeline or other automation tool.

class skilletlib.panoply.**Panoply**(*hostname: Optional[str] = None*, *api_username: Optional[str] = None*, *api_password: Optional[str] = None*, *api_port: Optional[int] = 443*, *serial_number: Optional[str] = None*, *debug: Optional[bool] = False*, *api_key: Optional[str] = None*)

Panoply is a wrapper around pan-python PanXAPI class to provide additional, commonly used functions

ad_hoc(*qs: str*) → str

Runs an ad_hoc command against this device.

example: type=op&action=complete&xpath=/operations/show/config/saved

Parameters **qs** – qs

Returns the unparsed xml document from the device API

backup_config()

Saves a named backup on the PAN-OS device. The format for the backup is 'config-backup-20190424000000.xml'

Returns xml results from the op command sequence

check_content_updates(*content_type: str) -> (<class 'str'>*, *None*)

Iterate through all available content of the specified type, locate and return the version with the highest version number. If that version is already installed, return None as no further action is necessary

Parameters **content_type** – type of content to check

Returns version-number to download and install or None if already at the latest

commit (*force_sync=True*) → str
Perform a commit operation on this device instance -

Raises **PanoplyException** – if commit failed

Parameters **force_sync** – Flag to enable sync commit or async

Returns String from the API indicating success or failure

commit_gpcs (*force_sync=True*) → str
Perform a commit operation on this device instance specifically for gpcs remote networks Note - you must do a full commit to panorama before you invoke this commit!

Raises **PanoplyException** – if commit failed

Parameters **force_sync** – Flag to enable sync commit or async

Returns String from the API indicating success or failure

connect (*allow_offline: Optional[bool] = False*) → None
Attempt to connect to this device instance

Parameters **allow_offline** – Do not raise an exception if this device is offline unless there is an authentication

error :return: None

deactivate_vm_license (*api_key: str = None*) → bool
Deactivate VM-Series Licenses. Will set the API Key is not already set.

Parameters **api_key** – Optional api_key.

Returns boolean True on success / False otherwise

execute_cli (*cmd_str: str*) → str
Short-cut to execute a simple CLI op cmd

Parameters **cmd_str** – CLI command to send to the NGFW

Returns raw output from the command

execute_cmd (*cmd: str*, *params: dict*, *context=None*) → str
Execute the given cmd using the xapi.

Parameters

- **cmd** – Valid options are: 'op', 'show', 'get', 'delete', 'set', 'edit', 'override', 'move', 'rename', 'clone', 'validate'
- **params** – valid parameters for the given cmd type
- **context** – skillet context

Returns raw results from the cmd output, raises SkilletLoaderException

execute_op (*cmd_str: str*, *cmd_xml=False*, *parse_result=True*) → str
Executes an 'op' command on the NGFW

Parameters

- **cmd_str** – op command to send
- **cmd_xml** – Flag to determine if op command requires XML encoding
- **parse_result** – Optional flag to indicate whether to return parsed xml results (xml_result) from xapi - not

all commands return valid XML. Setting this to 'false' will return the raw string from the API. :return: raw output from the device

fetch_license(*auth_code: str*, *force_fetch_license: bool = False*) → bool
Fetch and install licenses for PAN-OS NGFW

Parameters

- **auth_code** – Authorization code to use to license the NGFW
- **force_fetch_license** – Fetch licenses even if NGFW is already licensed

Returns True when license installation succeeds / False otherwise

filter_connected_devices(*filter_terms=None*) → list
Returns the list of connected devices filtered according to the given terms.

Filter terms are based on keys in the device facts. Matches are done using simple regex match

Parameters **filter_terms** – dict containing key value pairs. Keys match keys from the return device facts and values

are regex expressions used to match those keys :return: list of devices that match ALL filter terms

generate_baseline(*reset_hostname=False*) → str
Load baseline config that contains ONLY connecting username / password use device facts to determine which baseline template to load see template/panos/baseline_80.xml for example

Parameters

- **self** –
- **reset_hostname** – Flag to reset hostname back to a default value (baseline in this case)

Returns string contents of baseline config

generate_set_cli_from_configs(*previous_config: str*, *latest_config: str*) → list
Takes two configuration files, converts them to set commands, then returns only the commands found in the 'latest_config' vs the 'previous_config'. This allows the user to quickly configure one firewall, generate a 'set cli' diff and move those configs to another firewall

Parameters

- **previous_config** – Starting config
- **latest_config** – Ending config

Returns list of set cli commands required to convert previous to latest

generate_skillet(*from_candidate=False*) → list
Generates a skillet from the changes detected on this device. This will attempt to create the xml and xpaths for everything that is found to have changed

Parameters **from_candidate** – If your changes on in the candidate config, this will detect changes between the running

config and the candidate config. If False, this will detect changes between the running config and a generic baseline configuration :return: list of xpaths

generate_skillet_from_configs(*previous_config: str*, *latest_config: str*) → list
Generates a skillet from the diffs between two XML configuration files

Parameters

- **previous_config** – Configuration backup taken before desired changes are made to the device
- **latest_config** – Configuration backup taken after desired changes are made

Returns list of dictionaries that contain the following keys: * name * element * xpath * full_xpath

get_configuration(*config_source='running'*) → str
Get the configuration from the device.

Parameters **config_source** – Configuration source. This can be either 'candidate, running, baseline, or an

audit version number. Candidate is the candidate config, running is the running config, baseline is an autogenerated bare configuration, and version number is the previously saved running configuration. A positive version number will get the configuration version matching that id. A negative version number will get the most recent to least recent configuration versions where (-1) is the most recent previous running config. :return: configuration xml as a string or a blank string if not connected

get_configuration_version(*version: str*) → str
Returns a configuration version on the device. Use 'list_configuration_versions' to get a list of available options

Parameters **version** – version number of the configuration version to export

Returns configuration as an XML encoded string

get_extended_facts() → dict
Get extended facts about the device to which we are connected.

Return interfaces, security_rules, and zones

Returns dict with extended facts

get_facts() → dict
Gather system info and keep on self.facts This gets called on every connect

Returns dict containing all system facts

get_interfaces() → dict
Return a dict of all interfaces configured on this device. The dict has the following structure:

{

"ifnet": {

"entry": [

{ "name": "ethernet1/1", "zone": "internet", "fwd": "vr:default", "vsys": "1", "dyn-addr": null, "addr6": null, "tag": "0", "ip": "10.48.58.161/23", "id": "16", "addr": null

}

]

}, "hw": {

"entry": [

{ "name": "ethernet1/1", "duplex": "full", "type": "0", "state": "up", "st": "10000/full/up", "mac": "fa:16:3e:65:7d:05", "mode": "(autoneg)", "speed": "10000", "id": "16"

```
            }
        ]
    }
}
```

Returns dict with two keys 'ifnet' and 'hw'.

static get_ordered_xpaths() → tuple
Returns a list of ordered xpaths for use in ordering snippets and set commands

Will be enhanced one day with version and model specific information if necessary

Returns tuple of two lists, xpaths and post_xpaths

get_saved_configuration(*configuration_name: str*) → str
Returns a saved configuration on the device. Use 'list_saved_configuration' to get a list of available options

Parameters **configuration_name** – name of the saved configuration to export

Returns configuration as an XML encoded string

get_security_rules() → list
Return the list of configured security rules on this device.

- returns a blank list for Panorama devices!

Returns list of security rules

get_zones() → list
Return the list of configured zones on this device.

- returns a blank list for Panorama devices!

Returns list of zone names

has_running_jobs() → bool
Simple check to determine if there are any running jobs on this device

Returns bool True if there is currently a running job

import_file(*filename: str*, *file_contents: (<class 'str'>, <class 'bytes'>)*, *category: str*) → bool
Import the given file into this device

Parameters

- **filename** –
- **file_contents** –
- **category** – 'configuration'

Returns bool True on success

list_saved_configurations() → list
Returns a list of saved configuration files on this device

Returns list of saved configurations

load_baseline() → bool
Load baseline config that contains ONLY connecting username / password use device facts to determine which baseline template to load see template/panos/baseline_80.xml for example

Parameters **self** –

Returns bool true on success

load_config (*filename: str*) → bool
Loads the named configuration file into this device

Parameters **filename** – name of the configuration file on the device to load. Note this filename must already exist

on the target device :return: bool True on success

static sanitize_element (*element: str*) → str
Eliminate some unneeded characters from the XML snippet if they appear.

Parameters **element** – element str

Returns sanitized element str

set_at_path (*name: str*, *xpath: str*, *xml_str: str*) → None
Insert XML into the configuration tree at the specified xpath

Parameters

- **name** – name of the snippet - used in logging and debugging only
- **xpath** – full xpath where the xml element will be inserted
- **xml_str** – string representation of the XML element to insert

Returns None

set_license_api_key (*api_key: str*) → bool
Set's the Palo Alto Networks Support API Key in the firewall. This is required to deactivate a VM-Series NGFW.

Parameters **api_key** – Your support account API Key found on the support.paloaltonetworks.com site

Returns boolean True on success / False otherwise

update_dynamic_content (*content_type: str*) → bool
Check for newer dynamic content and install if found

Parameters **content_type** – type of content to check. can be either: 'content', 'anti-virus', 'wildfire'

Returns bool True on success

wait_for_device_ready (*interval=30*, *timeout=600*) → bool
Loop and wait until device is ready or times out

Parameters

- **interval** – how often to check in seconds
- **timeout** – how long to wait until we declare a timeout condition

Returns boolean true on ready, false on timeout

wait_for_job (*job_id: str*, *interval=10*, *timeout=600*) → bool
Loops until a given job id is completed. Will timeout after the timeout period if the device is offline or otherwise unavailable.

Parameters

- **job_id** – id the job to check and wait for
- **interval** – how long to wait between checks

- **timeout** – how long to wait with no response before we give up

Returns bool true on content updated, false otherwise

class skilletlib.panoply.**Panos**(*hostname: Optional[str], api_username: Optional[str], api_password: Optional[str], api_port: Optional[int] = 443, serial_number: Optional[str] = None, debug: Optional[bool] = False, api_key: Optional[str] = None*)

Panos is used to connect to PAN-OS devices that are expected to be currently and always online! Exceptions will be raised in any event that we cannot connect!

CHAPTER 6

Outputs

6.1 Executing Skillets

```
from skilletlib import SkilletLoader
sl = SkilletLoader()
skillet = sl.load_skillet('./ssl_decrypt_settings.skillet.yaml')
context = dict()
context['internal_zone'] = 'inside'
context['external_zone'] = 'outside'
out = skillet.execute(context)
print(out)
```

6.2 Examining the Output

Each Skillet type may return data differently. For example, *pan_validation* Skillets will return a dict with each key being the name of a snippet that was executed. It's value will be the results of that snippet.

```
{
    "update_schedule_configured": {
        "results": true,
        "label": "Ensure Update Schedules are Configured",
        "severity": "low",
        "documentation_link": "https://iron-skillet.readthedocs.io",
        "test": "update_schedule_object is not none",
        "output_message": "Snippet Validation Passed"
    },
}
```

6.3 Outputs Per Type

View the documentation for the 'get_results' method of each skillet class to determine what structure is returned by the Skillet type.

6.4 Output Templates

A very common use case is to collect some information from a NGFW, filter or otherwise manipulate that data, then display it to the user. This is so common, we've added a simple way to do both of these tasks in the same Skillet.

Output templates are normal jinja2 templates used to display data after the Skillet exeuciton is complete. By adding an 'output_template' key with a value of a relative path to a jinja2 template file, skilletlib will find and load that template, then render it using the outputs and context of the Skillet.

By default the template engine has access to the following context items:

- snippet_outputs
- captured_outputs
- context

See the 'output_template' directory in example_skillets for a complete example.

CHAPTER 7

Working with Objects

To make working with objects easier, we have created the following jinja filters:

- tag_present
- tag_absent
- attribute_present
- attribute_absent
- element_value
- element_value_contains
- append_uuid
- md5_hash

7.1 Jinja Filters

7.1.1 tag_present

This filter will validate that the given path is present in the variable object. The path argument is a '.' or '/' separated list. The variable object is inspected to verify each item in the path list is present. If all elements are found, this filter will return True. This is useful to validate a specific element is present.

```
- name: update_schedule_stats_service_configured
  label: Ensure Statistics Service is enabled
  test: update_schedule_object| tag_present('update-schedule.statistics-service')
```

7.1.2 element_value

This filter will return the value of the given path in the variable object. This value can then be used with any valid jinja expression. The path argument is a '.' or '/' separated list. The variable object is inspected to verify each item in the path list is present. If all elements are found, this filter will return leaf node text value.

```
- name: ensure_ip_address
  label: Ensure IP Address is configured
  test: device_system | element_value('ip-address') == '10.10.10.10'
```

7.1.3 attribute_present

This filter will determine if a node exists with the given attribute name and value. This is useful for parts of the configuration where there may be many 'entries' under a specific xpath. For example, security profiles or interfaces. This filter takes a configuration path as the first argument, similar to the tag_present filter. The second argument is the attribute name and the third is the attribute value. If the configuration path is found, and has an attribute that matches both the name and value, this filter will return True.

```
- name: check_profile_exists
  when: network_profiles is not none
  label: Ensure Named profile exists
  test: network_profiles | attribute_present('entry', 'name', 'default')
```

7.1.4 element_value_contains

This filter is useful for times when the xpath may contain a list of items. The path argument is a '.' or '/' separated list. The variable object is inspected to verify each item in the path list is present. If all elements are found, this filter will inspect the found value. If the value is a list, this filter will determine if the second argument is present in the list. If the value is a string, the filter will check if the string matches the second argument.

```
- name: security_rules_outbound_edl
  label: check for IronSkillet outbound EDL block rule member
  test: security_rule_outbound_edl | element_value_contains('destination.member',
↪'panw-bulletproof-ip-list')
  documentation_link: https://ironscotch.readthedocs.io/en/docs_dev/viz_guide_panos.
↪html#device-setup-telemetry-telemetry
```

7.1.5 items_present

This filter will iterate over a list and ensure all items from the first list appear in the second list. The second list can be a list of objects, in which case an optional path argument may be supplied.

Consider the case where you have a list of objects. Each object has it's own list of members. You want to ensure that all items from a list appear at least once in one of the objects member lists. For example, ensure a list of blocked application appear in at least one block rule.

```
snippets:
  - name: grab_security_rules
    cmd: parse
    variable: config
    outputs:
      - name: security_rules
```

(continues on next page)

(continued from previous page)

```
        capture_list: /config/devices/entry[@name='localhost.localdomain']/vsys/
↪entry[@name='vsys1']/rulebase/security/rules/entry
      - name: deny_rules
        capture_expression: security_rules
        filter_items: item | element_value('entry.action') == 'deny'
  - name: all_apps_blocked
    label: Ensure all blocked apps appear in the rules
    test: blocked_apps | items_present(deny_rules, 'entry.application.member')
    documentation_link: https://iron-skillet.readthedocs.io
```

CHAPTER 8

Additional Filters

Additional filters have been included from the Jinja2 Ansible Filters project.

Included filters

- b64decode
- b64encode
- basename
- bool
- checksum
- comment
- dirname
- expanduser
- expandvars
- extract
- fileglob
- flatten
- from_json
- from_yaml
- from_yaml_all
- ans_groupby
- hash
- mandatory
- md5
- quote

- ans_random
- random_mac
- realpath
- regex_escape
- regex_findall
- regex_replace
- regex_search
- relpath
- sha1
- shuffle
- splitext
- strftime
- subelements
- ternary
- to_datetime
- to_json
- to_nice_json
- to_nice_yaml
- to_uuid
- to_yaml
- type_debug
- win_basename
- win_dirname
- win_splitdrive

CHAPTER 9

About

Skilletlib is a library for working with Skillets. Skillets are a collection of configuration templates and some metadata about how those templates should be rendered and applied. Skillets were created to help manage complex shareable configuration sets for PAN-OS devices.

Skillets can also be thought of as wrappers around atomic automation units. A collection of PAN-OS XML configuration snippets can be grouped together as a unit to be shared and applied together. Skilletlib provides a convenient mechanism to examine and apply these automation units.

CHAPTER 10

Building Skillets

This documentation is intended document the internals the Skilletlib python library. For more information about building and using Configuration and Validation Skillets, refer to the Skillet Builder documentation.

CHAPTER 11

Disclaimer

This software is provided without support, warranty, or guarantee. Use at your own risk.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

H

I

L

N

P

R

S

T

U

W