

157 lines - 2 Removals

```

1 """
2     juliet_pipeline_steps: Implementation of pipeline steps to access the Juliet test suite and select testcases.
3 """
4
5 import logging
6 import typing
7
8 from pipeline_framework import PipelineState, PipelineStep
9
10
11 class EnumerateCWesPipelineStep(PipelineStep):
12     step_key = "enumerate_cwes"
13
14     def execute(self, input: dict, output: dict) -> None:
15         import juliet_access
16
17         output["cwes"] = juliet_access.enumerate_cwes()
18
19         if self.state.smoke_test:
20             self.logger.warning("Smoke test, only enumerating first 30 CWes")
21             output["cwes"] = output["cwes"][:30]
22
23     def output_ready(self, output: dict) -> bool:
24         return "cwes" in output.keys() and (len(output["cwes"]) == 118 or self.state.smoke_test)
25
26
27
28
29 class EnumerateTestcasesPipelineStep(PipelineStep):
30     step_key = "enumerate_testcases"
31
32     def input_ready(self, input: dict) -> bool:
33         return "cwes" in input.keys()
34
35     def execute(self, input: dict, output: dict) -> None:
36         import juliet_access
37
38         self.logger.debug(f'Enumerating testcases for {len(input["cwes"])} CWes')
39         output["testcases"] = sum(
40             [juliet_access.enumerate_testcases(cwe) for cwe in input["cwes"]], []
41         )
42

```

165 lines + 13 Additions

```

1 """
2     juliet_pipeline_steps: Implementation of pipeline steps to access the Juliet test suite and select testcases.
3 """
4
5 import logging
6 import typing
7
8 from pipeline_framework import PipelineState, PipelineStep
9
10
11 class EnumerateCWesPipelineStep(PipelineStep):
12     step_key = "enumerate_cwes"
13
14     def execute(self, input: dict, output: dict) -> None:
15         import juliet_access
16
17         if self.state.malware:
18             output["cwes"] = juliet_access.enumerate_cwes_malware(self.state.malware)
19         else:
20             output["cwes"] = juliet_access.enumerate_cwes(self.state.malware)
21
22         if self.state.smoke_test:
23             self.logger.warning("Smoke test, only enumerating first 30 CWes")
24             output["cwes"] = output["cwes"][:30]
25
26     def output_ready(self, output: dict) -> bool:
27         return "cwes" in output.keys() and (len(output["cwes"]) == 118 or self.state.smoke_test)
28
29
30
31
32 class EnumerateTestcasesPipelineStep(PipelineStep):
33     step_key = "enumerate_testcases"
34
35     def input_ready(self, input: dict) -> bool:
36         return "cwes" in input.keys()
37
38     def execute(self, input: dict, output: dict) -> None:
39         import juliet_access
40
41         self.logger.debug(f'Enumerating testcases for {len(input["cwes"])} CWes')
42         if self.state.malware:
43             output["testcases"] = sum(
44                 [juliet_access.enumerate_testcases_malware(cwe) for cwe in input["cwes"]], []
45             )
46         else:
47             output["testcases"] = sum(
48                 [juliet_access.enumerate_testcases(cwe) for cwe in input["cwes"]], []
49             )
50

```

```

43         if self.state.smoke_test:
44             self.logger.warning("Smoke test, only e
numerating random 1000 testcases")
45             shuffled_list = list(output["testcase
s"])
46             import random
47
48             random.shuffle(shuffled_list)
49             output["testcases"] = shuffled_list[:10
00]
50
51         def output_ready(self, output: dict) -> bool:
52             return "testcases" in output.keys() and (
53                 len(output["testcases"]) == 64099 or se
lf.state.smoke_test
54             )
55
56
57 class PrintTestcasesPerCWEPipelineStep(PipelineSte
p):
58     step_key = "print_testcases_per_cwe"
59
60     def __init__(
61         self,
62         config: dict,
63         state: PipelineState,
64         logger: logging.Logger,
65         comment: typing.Optional[str] = None,
66     ):
67         super().__init__(config, state, logger)
68
69         self.comment = comment
70
71     def input_ready(self, input: dict) -> bool:
72         return "cwes" in input.keys() and "testcase
s" in input.keys()
73
74     def execute(self, input: dict, output: dict) ->
None:
75         import utils
76
77         utils.print_testcases_per_cwe(input["cwe
s"], input["testcases"], self.comment)
78
79
80 class FilterTestcasesPipelineStep(PipelineStep):
81     step_key = "filter_testcases"
82
83     def __init__(
84         self,
85         config: dict,
86         state: PipelineState,
87         logger: logging.Logger,
88         function: str,
89         arguments: typing.Optional[list] = None,
90     ):
91         super().__init__(config, state, logger)
92
93         import juliet_access
94
95         if arguments is not None:
96             self.filter_fn = lambda cwes, testcase
s: juliet_access.filter_testcases(
97                 cwes,
98                 testcases,
99                 lambda testcase: getattr(juliet_acc
ess.filters, f"testcase_{function}") (
100                     testcase, **arguments

```

```

51         if self.state.smoke_test:
52             self.logger.warning("Smoke test, only e
numerating random 1000 testcases")
53             shuffled_list = list(output["testcase
s"])
54             import random
55
56             random.shuffle(shuffled_list)
57             output["testcases"] = shuffled_list[:10
0]
58
59         def output_ready(self, output: dict) -> bool:
60             return "testcases" in output.keys() and (
61                 len(output["testcases"]) == 64099 or se
lf.state.smoke_test
62             )
63
64
65 class PrintTestcasesPerCWEPipelineStep(PipelineSte
p):
66     step_key = "print_testcases_per_cwe"
67
68     def __init__(
69         self,
70         config: dict,
71         state: PipelineState,
72         logger: logging.Logger,
73         comment: typing.Optional[str] = None,
74     ):
75         super().__init__(config, state, logger)
76
77         self.comment = comment
78
79     def input_ready(self, input: dict) -> bool:
80         return "cwes" in input.keys() and "testcase
s" in input.keys()
81
82     def execute(self, input: dict, output: dict) ->
None:
83         import utils
84
85         utils.print_testcases_per_cwe(input["cwe
s"], input["testcases"], self.comment)
86
87
88 class FilterTestcasesPipelineStep(PipelineStep):
89     step_key = "filter_testcases"
90
91     def __init__(
92         self,
93         config: dict,
94         state: PipelineState,
95         logger: logging.Logger,
96         function: str,
97         arguments: typing.Optional[list] = None,
98     ):
99         super().__init__(config, state, logger)
100
101         import juliet_access
102
103         if arguments is not None:
104             self.filter_fn = lambda cwes, testcase
s: juliet_access.filter_testcases(
105                 cwes,
106                 testcases,
107                 lambda testcase: getattr(juliet_acc
ess.filters, f"testcase_{function}") (
108                     testcase, **arguments

```

```

101         ),
102     )
103     else:
104         self.filter_fn = lambda cwes, testcase
105             s: juliet_access.filter_testcases(
106                 cwes, testcases, getattr(juliet_acc
107                 ess.filters, f"testcase_{function}")
108             )
109     def input_ready(self, input: dict) -> bool:
110         return "testcases" in input.keys()
111     def execute(self, input: dict, output: dict) ->
112         None:
113         output["testcases"] = self.filter_fn(input
114         ["cwes"], input["testcases"])
115 class FilterCWESPipelineStep(PipelineStep):
116     step_key = "filter_cwes"
117     def __init__(
118         self,
119         config: dict,
120         state: PipelineState,
121         logger: logging.Logger,
122         function: str,
123         arguments: typing.Optional[list] = None,
124     ):
125         super().__init__(config, state, logger)
126         import juliet_access
127         if arguments is not None:
128             self.filter_fn = lambda cwes, testcase
129             s: juliet_access.filter_cwes(
130                 cwes,
131                 testcases,
132                 lambda cwe: getattr(juliet_access.f
133                 ilters, f"cwe_{function}") (
134                     cwe, **arguments
135                 ),
136             )
137         else:
138             self.filter_fn = lambda cwes, testcase
139             s: juliet_access.filter_cwes(
140                 cwes, testcases, getattr(juliet_acc
141                 ess.filters, f"cwe_{function}")
142             )
143     def input_ready(self, input: dict) -> bool:
144         return "cwes" in input.keys() and "testcase
145         s" in input.keys()
146     def execute(self, input: dict, output: dict) ->
147         None:
148         output["cwes"] = self.filter_fn(input["cwe
149         s"], input["testcases"])
150     __all__ = [
151         "EnumerateCWESPipelineStep",
152         "EnumerateTestcasesPipelineStep",
153         "PrintTestcasesPerCWEPipelineStep",
154         "FilterTestcasesPipelineStep",
155         "FilterCWESPipelineStep",
156     ]
157

```

```

109         ),
110     )
111     else:
112         self.filter_fn = lambda cwes, testcase
113             s: juliet_access.filter_testcases(
114                 cwes, testcases, getattr(juliet_acc
115                 ess.filters, f"testcase_{function}")
116             )
117     def input_ready(self, input: dict) -> bool:
118         return "testcases" in input.keys()
119     def execute(self, input: dict, output: dict) ->
120         None:
121         output["testcases"] = self.filter_fn(input
122         ["cwes"], input["testcases"])
123 class FilterCWESPipelineStep(PipelineStep):
124     step_key = "filter_cwes"
125     def __init__(
126         self,
127         config: dict,
128         state: PipelineState,
129         logger: logging.Logger,
130         function: str,
131         arguments: typing.Optional[list] = None,
132     ):
133         super().__init__(config, state, logger)
134         import juliet_access
135         if arguments is not None:
136             self.filter_fn = lambda cwes, testcase
137             s: juliet_access.filter_cwes(
138                 cwes,
139                 testcases,
140                 lambda cwe: getattr(juliet_access.f
141                 ilters, f"cwe_{function}") (
142                     cwe, **arguments
143                 ),
144             )
145         else:
146             self.filter_fn = lambda cwes, testcase
147             s: juliet_access.filter_cwes(
148                 cwes, testcases, getattr(juliet_acc
149                 ess.filters, f"cwe_{function}")
150             )
151     def input_ready(self, input: dict) -> bool:
152         return "cwes" in input.keys() and "testcase
153         s" in input.keys()
154     def execute(self, input: dict, output: dict) ->
155         None:
156         output["cwes"] = self.filter_fn(input["cwe
157         s"], input["testcases"])
158     __all__ = [
159         "EnumerateCWESPipelineStep",
160         "EnumerateTestcasesPipelineStep",
161         "PrintTestcasesPerCWEPipelineStep",
162         "FilterTestcasesPipelineStep",
163         "FilterCWESPipelineStep",
164     ]
165

```

