

169 lines - 3 Removals

```

1 """
2 juliet_access: Enumerate and access testcases f
   rom the Juliet suite.
3 """
4 import pathlib
5 import re
6 import typing
7
8 from . import filters, flow_variants
9
10
11 def enumerate_flow_variants() -> list[dict]:
12     """Enumerate all flow variants."""
13     return list(flow_variants.FLOW_VARIANT_DESCRIPTOR.values())
14
15

```

```

16 def enumerate_testcases(cwe: dict):
17     """Enumerate all testcases associated with the
   given CWE."""
18     cwe_path: pathlib.Path = cwe["Path"]

```

246 lines + 82 Additions

```

1 """
2 juliet_access: Enumerate and access testcases f
   rom the Juliet suite.
3 """
4 import pathlib
5 import re
6 import typing
7
8 from . import filters, flow_variants
9
10
11 def enumerate_flow_variants() -> list[dict]:
12     """Enumerate all flow variants."""
13     return list(flow_variants.FLOW_VARIANT_DESCRIPTOR.values())
14
15

```

```

16 def enumerate_testcases_malware(cwe: dict):
17
18     cwe_path: pathlib.Path = cwe["Path"]
19
20     # Collect object files
21     testcase_file_paths = []
22     for testcase_file_path in cwe_path.iterdir():
23         testcase_file_paths += [testcase_file_path]
24
25     testcase_files = {}
26     count = 1
27     for testcase_file_path in sorted(testcase_file_
   paths):
28         functional_variant = "MalVar-" + str(count)
29         count = count + 1
30         flow_variant = 1
31
32         key = (functional_variant, flow_variant)
33         if key not in testcase_files.keys():
34             testcase_files[key] = []
35
36         subfile_identifier = testcase_file_path.__s
   tr__().rsplit("_", 1)[-1]
37         testcase_files[key] += [
38             {"SubfileIdentifier": subfile_identifie
   r, "Path": testcase_file_path}
39         ]
40
41     testcases = [
42         {
43             "Weakness": cwe,
44             "FunctionalVariant": functional_varian
   t,
45             "FlowVariant": flow_variant,
46             "Files": v,
47         }
48         for (functional_variant, flow_variant), v i
   n testcase_files.items()
49     ]
50
51     return testcases
52
53

```

```

54 def enumerate_testcases(cwe: dict):
55     """Enumerate all testcases associated with the
   given CWE."""
56     cwe_path: pathlib.Path = cwe["Path"]

```

```

19
20 # Collect object files
21 testcase_file_regex = r"^CWE[0-9]+_+__([.])_([0
-9][0-9])([.])?\.\.+$"
22 testcase_file_paths = []
23 if (cwe_path / "s01").exists():
24     for split_path in cwe_path.iterdir():
25         for testcase_file_path in split_path.it
erdir():
26             if re.match(testcase_file_regex, tes
tcase_file_path.name) is not None:
27                 testcase_file_paths += [testcas
e_file_path]
28             # else:
29             #     print(f'Skipping {testcase_fil
e_path}')
30 else:
31     for testcase_file_path in cwe_path.iterdir
():
32         if re.match(testcase_file_regex, testca
se_file_path.name) is not None:
33             testcase_file_paths += [testcase_fi
le_path]
34         # else:
35         #     print(f'Skipping {testcase_file_pa
th}')
36
37 # Parse filenames (see https://samate.nist.gov/
SRD/resources/Juliet_Test_Suite_v1.2_for_C_Cpp_-_Us
er_Guide.pdf)
38 testcase_files = {}
39 for testcase_file_path in sorted(testcase_file_
paths):
40     matches = re.match(testcase_file_regex, tes
tcase_file_path.name)
41     functional_variant = matches[1]
42     flow_variant = int(matches[2])
43
44     key = (functional_variant, flow_variant)
45     if key not in testcase_files.keys():
46         testcase_files[key] = []
47
48     subfile_identifier = matches[3]
49     testcase_files[key] += [
50         {"SubfileIdentifier": subfile_identifie
r, "Path": testcase_file_path}
51     ]
52
53 testcases = [
54     {
55         "Weakness": cwe,
56         "FunctionalVariant": functional_varian
t,
57         "FlowVariant": flow_variant,
58         "Files": v,
59     }
60     for (functional_variant, flow_variant), v i
n testcase_files.items()
61 ]
62
63 return testcases
64
65
66 def _get_juliet_base_dir() -> pathlib.Path:

```

```

57
58 # Collect object files
59 testcase_file_regex = r"^CWE[0-9]+_+__([.])_([0
-9][0-9])([.])?\.\.+$"
60 testcase_file_paths = []
61 if (cwe_path / "s01").exists():
62     for split_path in cwe_path.iterdir():
63         for testcase_file_path in split_path.it
erdir():
64             if re.match(testcase_file_regex, te
stcase_file_path.name) is not None:
65                 testcase_file_paths += [testcas
e_file_path]
66             # else:
67             #     print(f'Skipping {testcase_fil
e_path}')
68 else:
69     for testcase_file_path in cwe_path.iterdir
():
70         if re.match(testcase_file_regex, testca
se_file_path.name) is not None:
71             testcase_file_paths += [testcase_fi
le_path]
72         # else:
73         #     print(f'Skipping {testcase_file_pa
th}')
74
75 # Parse filenames (see https://samate.nist.gov/
SRD/resources/Juliet_Test_Suite_v1.2_for_C_Cpp_-_Us
er_Guide.pdf)
76 testcase_files = {}
77 for testcase_file_path in sorted(testcase_file_
paths):
78     matches = re.match(testcase_file_regex, tes
tcase_file_path.name)
79     functional_variant = matches[1]
80     flow_variant = int(matches[2])
81
82     key = (functional_variant, flow_variant)
83     if key not in testcase_files.keys():
84         testcase_files[key] = []
85
86     subfile_identifier = matches[3]
87     testcase_files[key] += [
88         {"SubfileIdentifier": subfile_identifie
r, "Path": testcase_file_path}
89     ]
90
91 testcases = [
92     {
93         "Weakness": cwe,
94         "FunctionalVariant": functional_varian
t,
95         "FlowVariant": flow_variant,
96         "Files": v,
97     }
98     for (functional_variant, flow_variant), v i
n testcase_files.items()
99 ]
100
101 return testcases
102
103
104 def delete_linked_files(juliet_base_dir_env):
105     import os
106     try:

```

```

67     import os
68
69     try:
70         juliet_base_dir_env = os.environ["JULIET_BASE_DIR"]
71
72
73     except KeyError:
74         # Environment not set, try local share folder
75         local_share_folder = pathlib.Path.home() /
76             ".local" / "share"
77         if local_share_folder.exists():
78             # See if we have to make the ROMEO folder
79             local_share_romeo_folder = local_share_folder / "romeo"
80             local_share_romeo_folder.mkdir(exist_ok=True)
81             # Inside, there should be a "C" folder with the testcases
82             juliet_base_dir = local_share_romeo_folder / "C"
83             if not juliet_base_dir.exists():
84                 # If not, attempt to extract the object files
85                 tar_file = (
86                     pathlib.Path(__file__).parent.parent /
87                     "object_files" / "romeo_object_files.tar.gz"
88                 )
89                 if not tar_file.exists():
90                     raise ValueError(
91                         f"Attempted to extract object files from ROMEO distribution, but could not locate them at {str(tar_file)}"
92                     )
93

```

```

107         for filename in os.listdir(juliet_base_dir_env):
108             if filename.startswith("linked-"):
109                 file_path = os.path.join(juliet_base_dir_env, filename)
110                 os.remove(file_path)
111                 print(f"Deleted: {file_path}")
112             except FileNotFoundError:
113                 print(f"Directory '{juliet_base_dir_env}' not found.")
114             except PermissionError:
115                 print(f"Permission denied while trying to delete files in '{juliet_base_dir_env}'.")
116             except Exception as e:
117                 print(f"An error occurred: {e}")
118
119
120 def _get_juliet_base_dir(is_malware: bool) -> pathlib.Path:
121     import os
122
123     try:
124         if is_malware:
125             juliet_base_dir_env = os.environ["MALWARE_BASE_DIR"]
126             delete_linked_files(juliet_base_dir_env + "/dataset")
127         else:
128             juliet_base_dir_env = os.environ["JULIET_BASE_DIR"]
129         juliet_base_dir = pathlib.Path(juliet_base_dir_env)
130
131     except KeyError:
132         # Environment not set, try local share folder
133         local_share_folder = pathlib.Path.home() /
134             ".local" / "share"
135         if local_share_folder.exists():
136             # See if we have to make the ROMEO folder
137             local_share_romeo_folder = local_share_folder / "romeo"
138             local_share_romeo_folder.mkdir(exist_ok=True)
139             # Inside, there should be a "C" folder with the testcases
140             juliet_base_dir = local_share_romeo_folder / "C"
141             if not juliet_base_dir.exists():
142                 # If not, attempt to extract the object files
143                 tar_file = (
144                     pathlib.Path(__file__).parent.parent /
145                     "object_files" / "romeo_object_files.tar.gz"
146                 )
147                 if not tar_file.exists():
148                     raise ValueError(
149                         f"Attempted to extract object files from ROMEO distribution, but could not locate them at {str(tar_file)}"
150                     )
151

```

```

94
95         # Extract the file
96         import tarfile
97
98         tar_obj = tarfile.open(tar_file)
99         tar_obj.extractall(local_share_rome
100     o_folder)
101         tar_obj.close()
102     pass
103     else:
104         raise ValueError(
105             "JULIET_BASE_DIR environment variab
106         le not set and ~/.local/share does not exist"
107         )
108     return juliet_base_dir
109
110 def enumerate_cwes() -> list[dict]:
111     """Enumerate all CWEs in the Juliet test suite,
112     which can later be used to access testcases."""
113     juliet_base_dir = _get_juliet_base_dir()
114
115     assert str(juliet_base_dir.name).lower() == "c"
116     testcases = juliet_base_dir / "testcases"
117
118     cwe_regex = r"^CWE([0-9]+)(.+) $"
119     cwes = []
120
121     for cwe_path in testcases.iterdir():
122         cwe_path_basedir = cwe_path.name
123         assert (matches := re.match(cwe_regex, cwe_
124             path_basedir)) is not None
125
126         cwe = {}
127         cwe["WeaknessID"] = int(matches[1])
128         cwe["WeaknessShortenedName"] = str(matches
129             [2]).replace("_", " ")
130         cwe["Path"] = cwe_path
131         cwes += [cwe]
132
133     return list(sorted(cwes, key=lambda cwe: cwe["W
134         eaknessID"]))
135
136 def filter_testcases(
137     cwes: list[dict],

```

```

152
153         # Extract the file
154         import tarfile
155
156         tar_obj = tarfile.open(tar_file)
157         tar_obj.extractall(local_share_rome
158     o_folder)
159         tar_obj.close()
160     pass
161     else:
162         raise ValueError(
163             "JULIET_BASE_DIR environment variab
164         le not set and ~/.local/share does not exist"
165         )
166     return juliet_base_dir
167
168 def enumerate_cwes_malware(is_malware: bool) -> lis
169     t[dict]:
170     """Enumerate all CWEs in the Juliet test suite,
171     which can later be used to access testcases."""
172     testcases = _get_juliet_base_dir(is_malware)
173     malwares = []
174
175     for cwe_path in testcases.iterdir():
176         cwe_path_basedir = cwe_path.name
177
178         malware = {"WeaknessID": 1998, "WeaknessSho
179             rtenedName": "Mal-007", "Path": cwe_path}
180         malwares += [malware]
181
182     return list(sorted(malwares, key=lambda cwe: cw
183         e["WeaknessID"]))
184
185 def enumerate_cwes(is_malware: bool) -> list[dict]:
186     """Enumerate all CWEs in the Juliet test suite,
187     which can later be used to access testcases."""
188     juliet_base_dir = _get_juliet_base_dir(is_malwa
189         re)
190
191     assert str(juliet_base_dir.name).lower() == "c"
192     testcases = juliet_base_dir / "testcases"
193
194     cwe_regex = r"^CWE([0-9]+)(.+) $"
195     cwes = []
196
197     for cwe_path in testcases.iterdir():
198         cwe_path_basedir = cwe_path.name
199         assert (matches := re.match(cwe_regex, cwe_
200             path_basedir)) is not None
201
202         cwe = {}
203         cwe["WeaknessID"] = int(matches[1])
204         cwe["WeaknessShortenedName"] = str(matches
205             [2]).replace("_", " ")
206         cwe["Path"] = cwe_path
207         cwes += [cwe]
208
209     return list(sorted(cwes, key=lambda cwe: cwe["W
210         eaknessID"]))
211
212 def filter_testcases(
213     cwes: list[dict],

```

```

136     testcases: list[dict],
137     testcase_allowed: typing.Callable[[dict], boo
138     l],
139 ) -> list[dict]:
140     """Apply a given filter function to the testcas
141     es."""
142     allowed_testcases: list[dict] = []
143     for testcase in testcases:
144         if testcase_allowed(testcase):
145             allowed_testcases += [testcase]
146     return allowed_testcases
147
148 def filter_cwes(
149     cwes: list[dict],
150     testcases: list[dict],
151     cwe_allowed: typing.Callable[[dict, dict], boo
152     l],
153 ) -> list[dict]:
154     """Apply a given filter function to the cwe
155     s."""
156     allowed_cwes: list[dict] = []
157     for cwe in cwes:
158         if cwe_allowed(cwe, testcases):
159             allowed_cwes += [cwe]
160     return allowed_cwes
161
162 __all__ = [
163     "filters",
164     "filter_cwes",
165     "filter_testcases",
166     "enumerate_cwes",
167     "enumerate_testcases",
168 ]
169

```

```

211     testcases: list[dict],
212     testcase_allowed: typing.Callable[[dict], boo
213     l],
214 ) -> list[dict]:
215     """Apply a given filter function to the testcas
216     es."""
217     allowed_testcases: list[dict] = []
218     for testcase in testcases:
219         if testcase_allowed(testcase):
220             allowed_testcases += [testcase]
221     return allowed_testcases
222
223 def filter_cwes(
224     cwes: list[dict],
225     testcases: list[dict],
226     cwe_allowed: typing.Callable[[dict, dict], boo
227     l],
228 ) -> list[dict]:
229     """Apply a given filter function to the cwe
230     s."""
231     allowed_cwes: list[dict] = []
232     for cwe in cwes:
233         if cwe_allowed(cwe, testcases):
234             allowed_cwes += [cwe]
235     return allowed_cwes
236
237 __all__ = [
238     "filters",
239     "filter_cwes",
240     "filter_testcases",
241     "enumerate_cwes",
242     "enumerate_testcases",
243     "enumerate_cwes_malware",
244     "enumerate_testcases_malware",
245 ]
246

```