jupyter-server / **jupyter_server** Public

Code | Issues 193 | Pull requests 31 | Discussions | Actions | Projects

Commit 49b3439

Carreau authored yesterday · 38 / 38 · Verified

```
Move check origin into a util function and add it to websocket (#1630)
```

main (#1630) · v2.18.1 v2.18.0

1 parent e2e08c8 commit 49b3439

6 files changed +136 -40

Top

Filter files…

- jupyter_server
  - auth
    - login.py
  - base
    - handlers.py
    - websocket.py
  - utils.py
- tests
  - base
    - test_allow_origin_pat.py
  - test_utils.py

Search within code

jupyter_server/auth/login.py

```
@@ -10,6 +10,7 @@
from tornado.escape import url_escape

from ..base.handlers import JupyterHandler
+from ..utils import origin_matches_pat
from .decorator import allow_unauthenticated
from .security import passwd_check, set_password

@@ -73,7 +74,7 @@ def _redirect_safe(self, url, default=None):
                    if self.allow_origin:
                        allow = self.allow_origin == origin
                    elif self.allow_origin_pat:
-                       allow = self._origin_matches_pat(origin)
+                       allow = origin_matches_pat(self.allow_origin_pat, origin)
                if not allow:
                    # not allowed, use default
                    self.log.warning("Not allowing login redirect to %r" % url)
```

jupyter_server/base/handlers.py

```
@@ -36,6 +36,7 @@
    from jupyter_server.utils import (
        ensure_async,
        filefind,
+       origin_matches_pat,
        url_escape,
        url_is_absolute,
        url_path_join,
@@ -383,27 +384,6 @@ def allow_credentials(self) -> bool:
            """Whether to set Access-Control-Allow-Credentials"""
            return cast("bool", self.settings.get("allow_credentials", False))

-   def _origin_matches_pat(self, origin: str) -> bool:
-       """Check whether origin matches ``allow_origin_pat`` using full-string matching.
-
-       Uses ``re.fullmatch`` so the pattern must cover the entire origin string,
```

```
390      -         preventing prefix-bypass attacks (GHSA-24qx-w28j-9m6p/CVE-2026-40110).
391      -         Emits a warning in case a user relied on prefix-style patterns.
392      -         """
393      -         if not self.allow_origin_pat:
394      -             return False
395      -         if re.fullmatch(self.allow_origin_pat, origin):
396      -             return True
397      -         if re.match(self.allow_origin_pat, origin):
398      -             warnings.warn(
399      -                 f"allow_origin_pat {self.allow_origin_pat!r} only matched the request origin as a prefix. "
400      -                 "This has been replaced with a full string match. "
401      -                 "Update your pattern if you need to prefix-match the origin (e.g. append '.*')",
402      -                 UserWarning,
403      -                 stacklevel=2,
404      -             )
405      -         return False
406      -
407  387         def set_default_headers(self) -> None:
408  388             """Add CORS headers, if defined"""
409  389             super().set_default_headers()
         @@ -418,7 +398,7 @@ def set_cors_headers(self) -> None:
418  398                 self.set_header("Access-Control-Allow-Origin", self.allow_origin)
419  399             elif self.allow_origin_pat:
420  400                 origin = self.get_origin()
421      -             if origin and self._origin_matches_pat(origin):
     401 +             if origin and origin_matches_pat(self.allow_origin_pat, origin):
422  402                     self.set_header("Access-Control-Allow-Origin", origin)
423  403             elif self.token_authenticated and "Access-Control-Allow-Origin" not in self.settings.get(
424  404                 "headers", {}
         @@ -488,7 +468,7 @@ def check_origin(self, origin_to_satisfy_tornado: str = "") -> bool:
488  468             if self.allow_origin:
489  469                 allow = bool(self.allow_origin == origin)
490  470             elif self.allow_origin_pat:
491      -                 allow = self._origin_matches_pat(origin)
     471 +                 allow = origin_matches_pat(self.allow_origin_pat, origin)
492  472             else:
```

```
493  473                 # No CORS headers deny the request
494  474                 allow = False
     ↓↑  @@ -533,7 +513,7 @@ def check_referer(self) -> bool:
533  513             if self.allow_origin:
534  514                 allow = self.allow_origin == origin
535  515             elif self.allow_origin_pat:
536      -                 allow = self._origin_matches_pat(origin)
     516 +                 allow = origin_matches_pat(self.allow_origin_pat, origin)
537  517             else:
538  518                 # No CORS settings, deny the request
539  519                 allow = False
     ↓
```

jupyter_server/base/websocket.py

```
     ↑  @@ -8,7 +8,7 @@
8    8   from tornado.iostream import IOStream
9    9
10   10  from jupyter_server.base.handlers import JupyterHandler
11       - from jupyter_server.utils import JupyterServerAuthWarning
     11  + from jupyter_server.utils import JupyterServerAuthWarning, origin_matches_pat
12   12
13   13  # ping interval for keeping websockets alive (30 seconds)
14   14  WS_PING_INTERVAL = 30000
     ↓↑  @@ -71,7 +71,7 @@ def check_origin(self, origin: Optional[str] = None) ->
         bool:
71   71              if self.allow_origin:
72   72                  allow = self.allow_origin == origin
73   73              elif self.allow_origin_pat:
74       -                 allow = self._origin_matches_pat(origin)
     74  +                 allow = origin_matches_pat(self.allow_origin_pat, origin)
75   75              else:
76   76                  # No CORS headers deny the request
77   77                  allow = False
     ↓
```

jupyter_server/utils.py

```
     ↑  @@ -7,6 +7,7 @@
7    7   import errno
```

```
 8   8    import importlib.util
 9   9    import os
     10 + import re
10   11   import socket
11   12   import sys
12   13   import warnings
          @@ -42,6 +43,28 @@
42   43   ensure_async = _ensure_async
43   44
44   45
     46 + def origin_matches_pat(allow_origin_pat: str, origin: str) -> bool:
     47 +     """Check whether origin matches ``allow_origin_pat`` using full-string
          matching.
     48 +
     49 +     Uses ``re.fullmatch`` so the pattern must cover the entire origin string,
     50 +     preventing prefix-bypass attacks (GHSA-24qx-w28j-9m6p/CVE-2026-40110).
     51 +     Emits a warning in case a user relied on prefix-style patterns.
     52 +     """
     53 +     if not allow_origin_pat:
     54 +         return False
     55 +     if re.fullmatch(allow_origin_pat, origin):
     56 +         return True
     57 +     if re.match(allow_origin_pat, origin):
     58 +         warnings.warn(
     59 +             f"allow_origin_pat {allow_origin_pat!r} only matched the request
          origin as a prefix. "
     60 +             "This has been replaced with a full string match. "
     61 +             "Update your pattern if you need to prefix-match the origin (e.g.
          append '.*')",
     62 +             UserWarning,
     63 +             stacklevel=3,
     64 +         )
     65 +     return False
     66 +
     67 +
45   68   def url_path_join(*pieces: str) -> str:
46   69       """Join components of url into a relative url
47   70
```

tests/base/test_allow_origin_pat.py

```
@@ -7,6 +7,7 @@
to match only "https://trusted.example.com".
"""

+ import warnings
from unittest.mock import MagicMock

import pytest
@@ -45,7 +46,8 @@ def _make_handler(jp_serverapp, origin=None,
host="localhost:8888", referer=None

@pytest.mark.parametrize("origin", BYPASS_ORIGINS)
def test_check_origin_rejects_bypass(jp_serverapp, origin):
-    assert not _make_handler(jp_serverapp, origin=origin,
host="other.host:8888").check_origin()
+    with pytest.warns(UserWarning, match="allow_origin_pat.*"):
+        assert not _make_handler(jp_serverapp, origin=origin,
host="other.host:8888").check_origin()


def test_check_origin_allows_trusted(jp_serverapp):
@@ -55,29 +57,32 @@ def test_check_origin_allows_trusted(jp_serverapp):
@pytest.mark.parametrize("origin", BYPASS_ORIGINS)
def test_check_referer_rejects_bypass(jp_serverapp, origin):
    handler = _make_handler(jp_serverapp, host="other.host:8888", referer=f"
{origin}/page")
-    assert not handler.check_referer()
+    with pytest.warns(UserWarning, match="allow_origin_pat.*"):
+        assert not handler.check_referer()


@pytest.mark.parametrize("origin", BYPASS_ORIGINS)
def test_set_cors_headers_rejects_bypass(jp_serverapp, origin):
-    handler = _make_handler(jp_serverapp, origin=origin)
-    handler.set_cors_headers()
-    assert handler._headers.get("Access-Control-Allow-Origin") != origin
+    with pytest.warns(UserWarning, match="allow_origin_pat.*"):
+        handler = _make_handler(jp_serverapp, origin=origin)
```

```
        68  +         handler.set_cors_headers()
        69  +         assert handler._headers.get("Access-Control-Allow-Origin") != origin
66      70
67      71
68      72      @pytest.mark.parametrize("attacker_origin", BYPASS_ORIGINS)
69      73      def test_login_redirect_safe_rejects_bypass(jp_serverapp, attacker_origin):
70      74          request = HTTPRequest("GET", headers=HTTPHeaders({"Host":
                "localhost:8888"}))
71      75          request.connection = MagicMock()
72          -       handler = LoginHandler(jp_serverapp.web_app, request)
73          -       handler.settings["allow_origin"] = ""
74          -       handler.settings["allow_origin_pat"] = TRUSTED_PAT
        76  +       with pytest.warns(UserWarning, match="allow_origin_pat.*"):
        77  +           handler = LoginHandler(jp_serverapp.web_app, request)
        78  +           handler.settings["allow_origin"] = ""
        79  +           handler.settings["allow_origin_pat"] = TRUSTED_PAT
75      80
76          -       redirected_to = []
77          -       handler.redirect = lambda url, *a, **kw: redirected_to.append(url)
78          -       handler._redirect_safe(f"{attacker_origin}/capture",
                default=jp_serverapp.base_url)
        81  +       redirected_to = []
        82  +       handler.redirect = lambda url, *a, **kw: redirected_to.append(url)
        83  +       handler._redirect_safe(f"{attacker_origin}/capture",
                default=jp_serverapp.base_url)
79      84
80          -       assert redirected_to[0] != f"{attacker_origin}/capture"
        85  +       assert redirected_to[0] != f"{attacker_origin}/capture"
81      86
82      87
83      88      @pytest.mark.parametrize("attacker_origin", BYPASS_ORIGINS)
            ↕   @@ -96,7 +101,8 @@ class TestWSHandler(WebSocketMixin, JupyterHandler):
96      101         handler.settings["allow_origin_pat"] = TRUSTED_PAT
97      102         handler.skip_check_origin = lambda: False
98      103
99          -       assert not handler.check_origin(attacker_origin)
        104 +       with pytest.warns(UserWarning, match="allow_origin_pat.*"):
        105 +           assert not handler.check_origin(attacker_origin)
100     106
101     107
```

```
102  108  def test_truncated_pattern_warns_and_blocks(jp_serverapp):
```

tests/test_utils.py

```
     @@ -15,6 +15,7 @@
15   15       check_version,
16   16       filefind,
17   17       is_namespace_package,
     18   +   origin_matches_pat,
18   19       path2url,
19   20       run_sync_in_loop,
20   21       samefile_simple,
     @@ -164,3 +165,88 @@ def test_filefind(tmp_path, filename, result):
164  165       else:
165  166           with pytest.raises(result):
166  167               filefind(filename, [str(a), str(b)])
     168  +
     169  +
     170  + TRUSTED_PAT = r"https://trusted\.example\.com"
     171  +
     172  +
     173  + @pytest.mark.parametrize(
     174  +     "origin",
     175  +     [
     176  +         "https://trusted.example.com",
     177  +         # pattern is the full origin string
     178  +     ],
     179  + )
     180  + def test_origin_matches_pat_accepts_exact(origin):
     181  +     assert origin_matches_pat(TRUSTED_PAT, origin) is True
     182  +
     183  +
     184  + @pytest.mark.parametrize(
     185  +     "origin",
     186  +     [
     187  +         # suffix-bypass: pre-CVE-2026-40110 prefix matching would have allowed
           these
     188  +         "https://trusted.example.com.evil.com",
     189  +         "https://trusted.example.comedy",
```

```
190 +         "https://trusted.example.com:9999",
191 +         "https://trusted.example.com/path",
192 +         # newline injection — must not be allowed via $-anchor or otherwise
193 +         "https://trusted.example.com\nhttps://evil.com",
194 +     ],
195 + )
196 + def test_origin_matches_pat_rejects_full_match_failures(origin):
197 +     # These all match `re.match` (prefix) but NOT `re.fullmatch`, so the helper
198 +     # warns and returns False.
199 +     with pytest.warns(UserWarning, match="only matched the request origin as a
    prefix"):
200 +         assert origin_matches_pat(TRUSTED_PAT, origin) is False
201 +
202 +
203 + @pytest.mark.parametrize(
204 +     "origin",
205 +     [
206 +         "http://trusted.example.com",
207 +         "https://other.example.com",
208 +         "https://trusted.example.co",
209 +         "",
210 +     ],
211 + )
212 + def test_origin_matches_pat_rejects_non_match(origin):
213 +     # No prefix match either, so no warning.
214 +     with warnings.catch_warnings():
215 +         warnings.simplefilter("error")
216 +         assert origin_matches_pat(TRUSTED_PAT, origin) is False
217 +
218 +
219 + def test_origin_matches_pat_empty_pattern_rejects_all():
220 +     # Empty pattern must never allow any origin (empty-string regex would
    otherwise
221 +     # fullmatch only the empty string, but the early-return guard makes this
    explicit).
222 +     assert origin_matches_pat("", "") is False
223 +     assert origin_matches_pat("", "https://anything") is False
224 +
225 +
226 + def test_origin_matches_pat_respects_user_anchors():
```

```
227 +     # Patterns that already include ^ and $ should still work (fullmatch is idempotent
228 +     # with explicit anchors).
229 +     assert origin_matches_pat(r"^https://trusted\.example\.com$", "https://trusted.example.com")
230 +     assert not origin_matches_pat(
231 +         r"^https://trusted\.example\.com$", "https://trusted.example.com.evil"
232 +     )
233 +
234 +
235 + def test_origin_matches_pat_alternation():
236 +     pat = r"https://a\.com|https://b\.com"
237 +     assert origin_matches_pat(pat, "https://a.com") is True
238 +     assert origin_matches_pat(pat, "https://b.com") is True
239 +     # alternation must not be exploitable as a prefix:
240 +     with pytest.warns(UserWarning, match="only matched the request origin as a prefix"):
241 +         assert origin_matches_pat(pat, "https://a.comevil") is False
242 +
243 +
244 + def test_origin_matches_pat_unescaped_dot_is_a_footgun():
245 +     # Documented footgun: an operator who forgets to escape `.` writes a regex that
246 +     # treats it as a wildcard. The helper has no way to detect this — it is the
247 +     # caller's responsibility to escape literal characters in `allow_origin_pat`.
248 +     # This test pins the current behavior so a future change that auto-escapes or
249 +     # warns about unescaped dots is a deliberate, reviewed decision.
250 +     bad_pattern = r"https://trusted.example.com"  # dots NOT escaped
251 +     assert origin_matches_pat(bad_pattern, "https://trustedxexamplexcom") is True
252 +     assert origin_matches_pat(bad_pattern, "https://trusted-example-com") is True
```

Comments 0

Please sign in to comment.