jupyter-server / **jupyter_server** Public

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

Commit 057869a

Yann-P authored and **Carreau** committed yesterday · ✕ 7 / 38

```
Fix allow_origin_pat to do full matching instead of prefix matching
```

main · v2.18.1 v2.18.0

1 parent 4862199 commit 057869a

4 files changed +138 -6

Top

Filter files…

- jupyter_server
  - auth
    - login.py
  - base
    - handlers.py
    - websocket.py
- tests/base
  - test_allow_origin_pat.py

Search within code

jupyter_server/auth/login.py

```
@@ -68,7 +68,7 @@ def _redirect_safe(self, url, default=None):
68  68                  if self.allow_origin:
69  69                      allow = self.allow_origin == origin
70  70                  elif self.allow_origin_pat:
```

```
71      -                   allow = bool(re.match(self.allow_origin_pat, origin))
        71  +               allow = self._origin_matches_pat(origin)
72      72              if not allow:
73      73                  # not allowed, use default
74      74                  self.log.warning("Not allowing login redirect to %r" % url)
```

jupyter_server/base/handlers.py

```
@@ -383,6 +383,27 @@ def allow_credentials(self) -> bool:
383     383         """Whether to set Access-Control-Allow-Credentials"""
384     384         return cast("bool", self.settings.get("allow_credentials", False))
385     385
        386 +   def _origin_matches_pat(self, origin: str) -> bool:
        387 +       """Check whether origin matches ``allow_origin_pat`` using full-string matching.
        388 +
        389 +       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 +
386     407     def set_default_headers(self) -> None:
387     408         """Add CORS headers, if defined"""
388     409         super().set_default_headers()
```

```
@@ -397,7 +418,7 @@ def set_cors_headers(self) -> None:
397 418             self.set_header("Access-Control-Allow-Origin", self.allow_origin)
398 419         elif self.allow_origin_pat:
399 420             origin = self.get_origin()
400     -             if origin and re.match(self.allow_origin_pat, origin):
    421 +             if origin and self._origin_matches_pat(origin):
401 422                 self.set_header("Access-Control-Allow-Origin", origin)
402 423         elif self.token_authenticated and "Access-Control-Allow-Origin" not in
    self.settings.get(
403 424             "headers", {}
@@ -467,7 +488,7 @@ def check_origin(self, origin_to_satisfy_tornado: str =
    "") -> bool:
467 488         if self.allow_origin:
468 489             allow = bool(self.allow_origin == origin)
469 490         elif self.allow_origin_pat:
470     -             allow = bool(re.match(self.allow_origin_pat, origin))
    491 +             allow = self._origin_matches_pat(origin)
471 492         else:
472 493             # No CORS headers deny the request
473 494             allow = False
@@ -512,7 +533,7 @@ def check_referer(self) -> bool:
512 533         if self.allow_origin:
513 534             allow = self.allow_origin == origin
514 535         elif self.allow_origin_pat:
515     -             allow = bool(re.match(self.allow_origin_pat, origin))
    536 +             allow = self._origin_matches_pat(origin)
516 537         else:
517 538             # No CORS settings, deny the request
518 539             allow = False
```

jupyter_server/base/websocket.py

```
@@ -1,6 +1,5 @@
1 1 """Base websocket classes."""
2 2
3     - import re
4 3 import warnings
5 4 from typing import Optional, no_type_check
6 5 from urllib.parse import urlparse
```

```
@@ -72,7 +71,7 @@ def check_origin(self, origin: Optional[str] = None) ->
bool:
            if self.allow_origin:
                allow = self.allow_origin == origin
            elif self.allow_origin_pat:
-               allow = bool(re.match(self.allow_origin_pat, origin))
+               allow = self._origin_matches_pat(origin)
            else:
                # No CORS headers deny the request
                allow = False
```

tests/base/test_allow_origin_pat.py

```
@@ -0,0 +1,112 @@
+ """Tests for allow_origin_pat origin validation using re.fullmatch()
+
+ (GHSA-24qx-w28j-9m6p and CVE-2026-40110)
+
+ allow_origin_pat must match the full origin string. re.match() only anchors at
the
+ start, allowing "https://trusted.example.com.evil.com" to bypass a pattern
intended
+ to match only "https://trusted.example.com".
+ """
+
+ from unittest.mock import MagicMock
+
+ import pytest
+ from tornado.httpserver import HTTPRequest
+ from tornado.httputil import HTTPHeaders
+
+ from jupyter_server.auth.login import LoginHandler
+ from jupyter_server.base.handlers import JupyterHandler
+ from jupyter_server.base.websocket import WebSocketMixin
+
+ TRUSTED_PAT = r"https://trusted\.example\.com"
+ TRUSTED_ORIGIN = "https://trusted.example.com"
+
+ BYPASS_ORIGINS = [
+     "https://trusted.example.com.evil.com",
```

```
25  +     "https://trusted.example.comedy",
26  +     "https://trusted.example.com:9999",
27  + ]
28  +
29  +
30  + def _make_handler(jp_serverapp, origin=None, host="localhost:8888",
      referer=None):
31  +     headers = {"Host": host}
32  +     if origin:
33  +         headers["Origin"] = origin
34  +     if referer:
35  +         headers["Referer"] = referer
36  +     request = HTTPRequest("GET", headers=HTTPHeaders(headers))
37  +     request.connection = MagicMock()
38  +     handler = JupyterHandler(jp_serverapp.web_app, request)
39  +     handler.settings["allow_origin"] = ""
40  +     handler.settings["allow_origin_pat"] = TRUSTED_PAT
41  +     # skip_check_origin() requires async auth context; bypass it to reach
      pattern matching
42  +     handler.skip_check_origin = lambda: False
43  +     return handler
44  +
45  +
46  + @pytest.mark.parametrize("origin", BYPASS_ORIGINS)
47  + def test_check_origin_rejects_bypass(jp_serverapp, origin):
48  +     assert not _make_handler(jp_serverapp, origin=origin,
      host="other.host:8888").check_origin()
49  +
50  +
51  + def test_check_origin_allows_trusted(jp_serverapp):
52  +     assert _make_handler(jp_serverapp, origin=TRUSTED_ORIGIN,
      host="other.host:8888").check_origin()
53  +
54  +
55  + @pytest.mark.parametrize("origin", BYPASS_ORIGINS)
56  + def test_check_referer_rejects_bypass(jp_serverapp, origin):
57  +     handler = _make_handler(jp_serverapp, host="other.host:8888", referer=f"
      {origin}/page")
58  +     assert not handler.check_referer()
59  +
```

```
60  +
61  + @pytest.mark.parametrize("origin", BYPASS_ORIGINS)
62  + def test_set_cors_headers_rejects_bypass(jp_serverapp, origin):
63  +     handler = _make_handler(jp_serverapp, origin=origin)
64  +     handler.set_cors_headers()
65  +     assert handler._headers.get("Access-Control-Allow-Origin") != origin
66  +
67  +
68  + @pytest.mark.parametrize("attacker_origin", BYPASS_ORIGINS)
69  + def test_login_redirect_safe_rejects_bypass(jp_serverapp, attacker_origin):
70  +     request = HTTPRequest("GET", headers=HTTPHeaders({"Host":
      "localhost:8888"}))
71  +     request.connection = MagicMock()
72  +     handler = LoginHandler(jp_serverapp.web_app, request)
73  +     handler.settings["allow_origin"] = ""
74  +     handler.settings["allow_origin_pat"] = TRUSTED_PAT
75  +
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)
79  +
80  +     assert redirected_to[0] != f"{attacker_origin}/capture"
81  +
82  +
83  + @pytest.mark.parametrize("attacker_origin", BYPASS_ORIGINS)
84  + def test_websocket_check_origin_rejects_bypass(jp_serverapp, attacker_origin):
85  +     request = HTTPRequest(
86  +         "GET",
87  +         headers=HTTPHeaders({"Origin": attacker_origin, "Host":
      "other.host:8888"}),
88  +     )
89  +     request.connection = MagicMock()
90  +
91  +     class TestWSHandler(WebSocketMixin, JupyterHandler):
92  +         pass
93  +
94  +     handler = TestWSHandler(jp_serverapp.web_app, request)
95  +     handler.settings["allow_origin"] = ""
96  +     handler.settings["allow_origin_pat"] = TRUSTED_PAT
```

```
97  +     handler.skip_check_origin = lambda: False
98  +
99  +     assert not handler.check_origin(attacker_origin)
100 +
101 +
102 + def test_truncated_pattern_warns_and_blocks(jp_serverapp):
103 +     """Prefix-only pattern (e.g. 'https://trusted') emits UserWarning and
      blocks the request."""
104 +     handler = _make_handler(
105 +         jp_serverapp, origin="https://trusted.example.com",
      host="other.host:8888"
106 +     )
107 +     handler.settings["allow_origin_pat"] = r"https://trusted"
108 +
109 +     with pytest.warns(UserWarning, match="only matched the request origin as a
      prefix"):
110 +         result = handler.check_origin()
111 +
112 +     assert not result
```

Comments 0

Please sign in to comment.