

jupyter-server / jupyter_server Public

<> Code Issues 192 Pull requests 31 Discussions Actions Projects

Commit 057869a



Yann-P authored and Carreau committed 3 days ago · 7 / 38



Fix allow_origin_pat to do full matching instead of prefix matching

main · v2.18.2 ... 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
388 +     matching.
389 +     Uses ``re.fullmatch`` so the pattern must cover the entire origin
390 +     string,
391 +     preventing prefix-bypass attacks (GHSA-24qx-w28j-9m6p/CVE-2026-40110).
392 +     Emits a warning in case a user relied on prefix-style patterns.
393 +     """
394 +     if not self.allow_origin_pat:
395 +         return False
396 +     if re.fullmatch(self.allow_origin_pat, origin):
397 +         return True
398 +     if re.match(self.allow_origin_pat, origin):
399 +         warnings.warn(
400 +             f"allow_origin_pat {self.allow_origin_pat!r} only matched the
401 +             request origin as a prefix. "
402 +             "This has been replaced with a full string match. "
403 +             "Update your pattern if you need to prefix-match the origin
404 +             (e.g. append '.*')",
405 +             UserWarning,
406 +             stacklevel=2,
407 +         )
408 +         return False
409 +
410 def set_default_headers(self) -> None:
411     """Add CORS headers, if defined"""
412     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:
72 71         if self.allow_origin:
73 72             allow = self.allow_origin == origin
74 73         elif self.allow_origin_pat:
75 -             allow = bool(re.match(self.allow_origin_pat, origin))
74 +             allow = self._origin_matches_pat(origin)
76 75         else:
77 76             # No CORS headers deny the request
78 77             allow = False

```

```

tests/base/test_allow_origin_pat.py
... @@ -0,0 +1,112 @@
1 + """Tests for allow_origin_pat origin validation using re.fullmatch()
2 +
3 + (GHSA-24qx-w28j-9m6p and CVE-2026-40110)
4 +
5 + allow_origin_pat must match the full origin string. re.match() only anchors at
6 + start, allowing "https://trusted.example.com.evil.com" to bypass a pattern
7 + intended
8 + to match only "https://trusted.example.com".
9 + """
10 + from unittest.mock import MagicMock
11 +
12 + import pytest
13 + from tornado.httpserver import HTTPRequest
14 + from tornado.httputil import HTTPHeaders
15 +
16 + from jupyter_server.auth.login import LoginHandler
17 + from jupyter_server.base.handlers import JupyterHandler
18 + from jupyter_server.base.websocket import WebSocketMixin
19 +
20 + TRUSTED_PAT = r"https://trusted\.example\.com"
21 + TRUSTED_ORIGIN = "https://trusted.example.com"
22 +
23 + BYPASS_ORIGINS = [
24 +     "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",
31 +                  referer=None):
32 +     headers = {"Host": host}
33 +     if origin:
34 +         headers["Origin"] = origin
35 +     if referer:
36 +         headers["Referer"] = referer
37 +     request = HTTPRequest("GET", headers=HTTPHeaders(headers))
38 +     request.connection = MagicMock()
39 +     handler = JupyterHandler(jp_serverapp.web_app, request)
40 +     handler.settings["allow_origin"] = ""
41 +     handler.settings["allow_origin_pat"] = TRUSTED_PAT
42 +     # skip_check_origin() requires async auth context; bypass it to reach
43 +     # pattern matching
44 +     handler.skip_check_origin = lambda: False
45 +     return handler
46 +
47 + @pytest.mark.parametrize("origin", BYPASS_ORIGINS)
48 + def test_check_origin_rejects_bypass(jp_serverapp, origin):
49 +     assert not _make_handler(jp_serverapp, origin=origin,
50 +                             host="other.host:8888").check_origin()
51 +
52 + def test_check_origin_allows_trusted(jp_serverapp):
53 +     assert _make_handler(jp_serverapp, origin=TRUSTED_ORIGIN,
54 +                         host="other.host:8888").check_origin()
55 +
56 + @pytest.mark.parametrize("origin", BYPASS_ORIGINS)
57 + def test_check_referer_rejects_bypass(jp_serverapp, origin):
58 +     handler = _make_handler(jp_serverapp, host="other.host:8888", referer=f"
59 +                             {origin}/page")
60 +     assert not handler.check_referer()
```

```
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=HTTPHeaderDict({"Host":
71 +         "localhost:8888"}))
72 +     request.connection = MagicMock()
73 +     handler = LoginHandler(jp_serverapp.web_app, request)
74 +     handler.settings["allow_origin"] = ""
75 +     handler.settings["allow_origin_pat"] = TRUSTED_PAT
76 +
77 +     redirected_to = []
78 +     handler.redirect = lambda url, *a, **kw: redirected_to.append(url)
79 +     handler._redirect_safe(f"{attacker_origin}/capture",
80 +         default=jp_serverapp.base_url)
81 +
82 +     assert redirected_to[0] != f"{attacker_origin}/capture"
83 +
84 + @pytest.mark.parametrize("attacker_origin", BYPASS_ORIGINS)
85 + def test_websocket_check_origin_rejects_bypass(jp_serverapp, attacker_origin):
86 +     request = HTTPRequest(
87 +         "GET",
88 +         headers=HTTPHeaderDict({"Origin": attacker_origin, "Host":
89 +             "other.host:8888"}),
90 +     )
91 +     request.connection = MagicMock()
92 +
93 +     class TestWSHandler(WebSocketMixin, JupyterHandler):
94 +         pass
95 +
96 +     handler = TestWSHandler(jp_serverapp.web_app, request)
97 +     handler.settings["allow_origin"] = ""
98 +     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
104 +     blocks the request."""
105 +     handler = _make_handler(
106 +         jp_serverapp, origin="https://trusted.example.com",
107 +         host="other.host:8888"
108 +     )
109 +     handler.settings["allow_origin_pat"] = r"https://trusted"
110 +
111 +     with pytest.warns(UserWarning, match="only matched the request origin as a
112 +     prefix"):
113 +         result = handler.check_origin()
114 +
115 +     assert not result
```

Comments 0



Please [sign in](#) to comment.