PrefectHQ / **prefect** Public

Code Issues 818 Pull requests 39 Discussions Actions Security an

⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Commit `0d3ab3c`

devin-ai-integration[bot] and **desertaxle** committed on Jan 23 · ✓ 58 / 62

```
Make /api/events/in backward compatible when auth is not configured

When PREFECT_SERVER_API_AUTH_STRING is not set:
- Server accepts connections without requiring 'prefect' subprotocol or auth handshake
- Client connects without sending auth messages

When PREFECT_SERVER_API_AUTH_STRING is set:
- Server requires 'prefect' subprotocol and auth handshake
- Client sends auth messages with the configured token

This maintains backward compatibility for servers without auth while
fixing the security vulnerability when auth is configured.

Co-Authored-By: alex.s@prefect.io <ajstreed1@gmail.com>
```

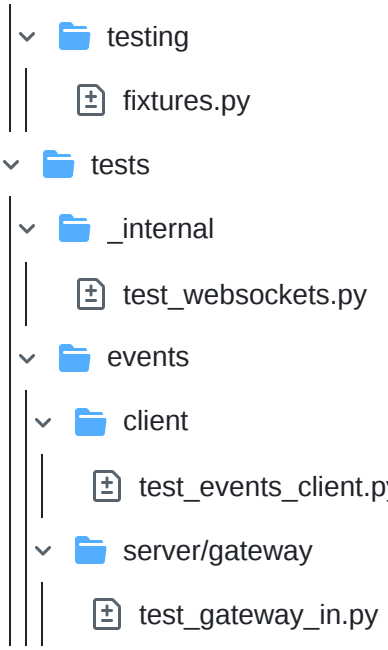
1 parent a0f2938 commit 0d3ab3c

6 files changed +92 -87

Top

Filter files…

- src/prefect
  - events
    - clients.py
  - server/api
    - events.py



Search within code

src/prefect/events/clients.py

```
@@ -284,10 +284,14 @@ def __init__(
284 284
285 285             self._auth_token = PREFECT_API_AUTH_STRING.value()
286 286             self._events_socket_url = events_in_socket_from_api_url(api_url)
287     -         self._connect = websocket_connect(
288     -             self._events_socket_url,
289     -             subprotocols=[Subprotocol("prefect")],
290     -         )
    287 +         # Only use the prefect subprotocol when auth is configured
    288 +         if self._auth_token:
    289 +             self._connect = websocket_connect(
    290 +                 self._events_socket_url,
    291 +                 subprotocols=[Subprotocol("prefect")],
    292 +             )
    293 +         else:
    294 +             self._connect = websocket_connect(self._events_socket_url)
291 295             self._websocket = None
292 296             self._reconnection_attempts = reconnection_attempts
293 297             self._unconfirmed_events = []
@@ -379,28 +383,31 @@ async def _reconnect(self) -> None:
379 383                 )
```

```
380  384                raise
381  385
382       -          # Authenticate with the server
383       -          logger.debug("Authenticating...")
384       -          await self._websocket.send(
385       -              orjson.dumps({"type": "auth", "token": self._auth_token}).decode()
386       -          )
387       -
388       -          try:
389       -              message: Dict[str, Any] = orjson.loads(await
          self._websocket.recv())
390       -              logger.debug("Auth result: %s", message)
391       -              assert message["type"] == "auth_success", message.get("reason", "")
392       -          except AssertionError as e:
393       -              raise Exception(
394       -                  "Unable to authenticate to the event stream. Please ensure the
          "
395       -                  "provided auth_token you are using is valid for this
          environment. "
396       -                  f"Reason: {e.args[0]}"
          386  +          # Authenticate with the server only when auth is configured
          387  +          if self._auth_token:
          388  +              logger.debug("Authenticating...")
          389  +              await self._websocket.send(
          390  +                  orjson.dumps({"type": "auth", "token":
          self._auth_token}).decode()
397  391                )
398       -          except ConnectionClosedError as e:
399       -              reason = getattr(e.rcvd, "reason", None)
400       -              msg = "Unable to authenticate to the event stream. Please ensure
          the "
401       -              msg += "provided auth_token you are using is valid for this
          environment. "
402       -              msg += f"Reason: {reason}" if reason else ""
403       -              raise Exception(msg) from e
          392  +
          393  +              try:
          394  +                  message: Dict[str, Any] = orjson.loads(await
          self._websocket.recv())
          395  +                  logger.debug("Auth result: %s", message)
```

```
396 +                 assert message["type"] == "auth_success", message.get("reason",
    "")
397 +             except AssertionError as e:
398 +                 raise Exception(
399 +                     "Unable to authenticate to the event stream. Please ensure
    the "
400 +                     "provided auth_token you are using is valid for this
    environment. "
401 +                     f"Reason: {e.args[0]}"
402 +                 )
403 +             except ConnectionClosedError as e:
404 +                 reason = getattr(e.rcvd, "reason", None)
405 +                 msg = "Unable to authenticate to the event stream. Please
    ensure the "
406 +                 msg += (
407 +                     "provided auth_token you are using is valid for this
    environment. "
408 +                 )
409 +                 msg += f"Reason: {reason}" if reason else ""
410 +                 raise Exception(msg) from e
404 411
405 412             events_to_resend = self._unconfirmed_events
406 413             logger.debug("Resending %s unconfirmed events.", len(events_to_resend))
```

src/prefect/server/api/events.py

```
    @@ -33,6 +33,7 @@
33  33  from prefect.settings import (
34  34      PREFECT_EVENTS_MAXIMUM_WEBSOCKET_BACKFILL,
35  35      PREFECT_EVENTS_WEBSOCKET_BACKFILL_PAGE_SIZE,
    36 +    get_current_settings,
36  37  )
37  38
38  39  if TYPE_CHECKING:
    @@ -65,9 +66,22 @@ async def create_events(
65  66  async def stream_events_in(websocket: WebSocket) -> None:
66  67      """Open a WebSocket to stream incoming Events"""
67  68
68      -    websocket = await subscriptions.accept_prefect_socket(websocket)
```

```
69      -     if not websocket:
70      -         return
        69    +     # Check if authentication is configured
        70    +     auth_setting = (
        71    +         auth_setting_secret.get_secret_value()
        72    +         if (auth_setting_secret :=
                  get_current_settings().server.api.auth_string)
        73    +         else None
        74    +     )
        75    +
        76    +     if auth_setting:
        77    +         # When auth is configured, require the prefect subprotocol and auth
                  handshake
        78    +         websocket = await subscriptions.accept_prefect_socket(websocket)
        79    +         if not websocket:
        80    +             return
        81    +     else:
        82    +         # When auth is not configured, accept connections without requiring
        83    +         # the prefect subprotocol or auth handshake for backward compatibility
        84    +         await websocket.accept()
71      85
72      86        try:
73      87            async with messaging.create_event_publisher() as publisher:
```

src/prefect/testing/fixtures.py

```
@@ -341,20 +341,23 @@ async def handler(socket: ServerConnection) -> None:
341     341               await outgoing_events(socket)
342     342
343     343       async def incoming_events(socket: ServerConnection):
344     -         # 1. authentication
345     -         auth_message = json.loads(await socket.recv())
346     -
347     -         assert auth_message["type"] == "auth"
348     -         recorder.token = auth_message["token"]
349     -         if puppeteer.token is not None and puppeteer.token != recorder.token:
350     -             if not puppeteer.hard_auth_failure:
351     -                 await socket.send(
352     -                     json.dumps({"type": "auth_failure", "reason": "nope"})
353     -                 )
```

```
354      -             await socket.close(WS_1008_POLICY_VIOLATION)
355      -             return
         344 +         # 1. authentication (only when puppeteer.token is set, indicating auth is expected)
         345 +         # This mirrors the server behavior where auth is only required when
         346 +         # PREFECT_SERVER_API_AUTH_STRING is configured
         347 +         if puppeteer.token is not None:
         348 +             auth_message = json.loads(await socket.recv())
         349 +
         350 +             assert auth_message["type"] == "auth"
         351 +             recorder.token = auth_message["token"]
         352 +             if puppeteer.token != recorder.token:
         353 +                 if not puppeteer.hard_auth_failure:
         354 +                     await socket.send(
         355 +                         json.dumps({"type": "auth_failure", "reason": "nope"})
         356 +                     )
         357 +                 await socket.close(WS_1008_POLICY_VIOLATION)
         358 +                 return
356      359
357      -         await socket.send(json.dumps({"type": "auth_success"}))
         360 +         await socket.send(json.dumps({"type": "auth_success"}))
358      361
359      362           # 2. receive events
360      363           while True:
```

tests/_internal/test_websockets.py

```
         ...   @@ -1,10 +1,8 @@
1        -   import json
2        1     import ssl
3        2     import warnings
4        3     from unittest.mock import patch
5        4
6        5     from websockets.asyncio.client import connect
7        -   from websockets.protocol import Subprotocol
8        6
9        7     from prefect._internal.websockets import (
10       8         create_ssl_context_for_websocket,
         ...   @@ -166,17 +164,11 @@ async def
              test_websocket_custom_headers_with_websocket_connect(hosted_api_server
```

```
166  164          custom_headers = {"X-Custom-Header": "test-value"}
167  165
168  166          with temporary_settings({PREFECT_CLIENT_CUSTOM_HEADERS: custom_headers}):
169       -          connector = websocket_connect(
170       -              events_in_socket_from_api_url(hosted_api_server),
171       -              subprotocols=[Subprotocol("prefect")],
172       -          )
     167  +          connector =
          websocket_connect(events_in_socket_from_api_url(hosted_api_server))
173  168              # Make sure we can connect to the websocket successfully with the
          custom headers
     169  +          # Note: When PREFECT_SERVER_API_AUTH_STRING is not set, the /events/in
          endpoint
     170  +          # accepts connections without requiring the "prefect" subprotocol or
          auth handshake
174  171              async with connector as websocket:
175  172                  pong = await websocket.ping()
176  173                  await pong
177       -
178       -              # Complete the auth handshake required by /events/in endpoint
179       -              await websocket.send(json.dumps({"type": "auth", "token": None}))
180       -              auth_response = json.loads(await websocket.recv())
181       -              assert auth_response["type"] == "auth_success"
182  174              # If we get here, the connection worked with custom headers
```

tests/events/client/test_events_client.py

```
          @@ -524,10 +524,6 @@ async def ping(self):
524  524              async def send(self, data):
525  525                  pass
526  526
527       -          async def recv(self):
528       -              # Return auth_success response for the auth handshake
529       -              return '{"type": "auth_success"}'
530       -
531  527          class MockConnect:
532  528              def __init__(self):
533  529                  self.connection = None
          @@ -547,7 +543,7 @@ def mock_connect(*args, **kwargs):
547  543
```

```
548  544          monkeypatch.setattr("prefect.events.clients.websocket_connect",
                  mock_connect)
549  545
550       -      # Should succeed after retrying
     546  +      # Should succeed after retrying (no auth required when
                  PREFECT_API_AUTH_STRING is not set)
551  547          async with PrefectEventsClient("ws://localhost") as client:
552  548              assert client._websocket is not None
553  549
                  @@ -641,10 +637,6 @@ async def ping(self):
641  637              async def send(self, data):
642  638                  pass
643  639
644       -          async def recv(self):
645       -              # Return auth_success response for the auth handshake
646       -              return '{"type": "auth_success"}'
647       -
648  640          class MockConnect:
649  641              def __init__(self):
650  642                  self.connection = None
                  @@ -664,6 +656,7 @@ def mock_connect(*args, **kwargs):
664  656
665  657          monkeypatch.setattr("prefect.events.clients.websocket_connect",
                  mock_connect)
666  658
     659  +      # No auth required when PREFECT_API_AUTH_STRING is not set
667  660          async with PrefectEventsClient("ws://localhost") as client:
668  661              assert client._websocket is not None
669  662
```

...ts/events/server/gateway/test_gateway_in.py

```
                  @@ -47,29 +47,33 @@ async def write_events(monkeypatch: pytest.MonkeyPatch):
47   47          return mock_write_events
48   48
49   49
50        - def test_streaming_requires_prefect_subprotocol(
     50   + def test_streaming_requires_prefect_subprotocol_when_auth_configured(
51   51          test_client: TestClient,
```

```
52  52      ):
53          -     with pytest.raises(WebSocketDisconnect) as exception:
54          -         with test_client.websocket_connect("/api/events/in", subprotocols=[]):
55          -             pass
        53  +     """When PREFECT_SERVER_API_AUTH_STRING is set, the prefect subprotocol is
                  required."""
        54  +     with temporary_settings(updates={PREFECT_SERVER_API_AUTH_STRING: "valid-
                  token"}):
        55  +         with pytest.raises(WebSocketDisconnect) as exception:
        56  +             with test_client.websocket_connect("/api/events/in", subprotocols=
                          []):
        57  +                 pass
56  58
57          -     assert exception.value.code == WS_1002_PROTOCOL_ERROR
        59  +     assert exception.value.code == WS_1002_PROTOCOL_ERROR
58  60
59  61
60          - def test_streaming_requires_authentication(
        62  + def test_streaming_requires_authentication_when_auth_configured(
61  63          test_client: TestClient,
62  64          event1: Event,
63  65      ):
64          -     with pytest.raises(WebSocketDisconnect) as exception:
65          -         with test_client.websocket_connect(
66          -             "/api/events/in", subprotocols=["prefect"]
67          -         ) as websocket:
68          -             websocket.send_text(event1.model_dump_json())
69          -             websocket.receive_text()
        66  +     """When PREFECT_SERVER_API_AUTH_STRING is set, an auth message is
                  required."""
        67  +     with temporary_settings(updates={PREFECT_SERVER_API_AUTH_STRING: "valid-
                  token"}):
        68  +         with pytest.raises(WebSocketDisconnect) as exception:
        69  +             with test_client.websocket_connect(
        70  +                 "/api/events/in", subprotocols=["prefect"]
        71  +             ) as websocket:
        72  +                 websocket.send_text(event1.model_dump_json())
        73  +                 websocket.receive_text()
70  74
71          -     assert exception.value.code == WS_1008_POLICY_VIOLATION
```

```
72       -      assert exception.value.reason == "Expected 'auth' message"
         75     +      assert exception.value.code == WS_1008_POLICY_VIOLATION
         76     +      assert exception.value.reason == "Expected 'auth' message"
73       77
74       78
75       79     def test_streaming_rejects_invalid_token(
                ↓↑  @@ -109,25 +113,17 @@ def test_streaming_rejects_missing_token(
109      113            assert exception.value.reason == "Auth required but no token provided"
110      114
111      115
112      -      def test_stream_events_in(
         116    + def test_stream_events_in_without_auth(
113      117        test_client: TestClient,
114      118        frozen_time: DateTime,
115      119        event1: Event,
116      120        event2: Event,
117      121        stream_publish: mock.AsyncMock,
118      122    ):
         123    +     """When PREFECT_SERVER_API_AUTH_STRING is not set, connections work without auth."""
119      124        websocket: WebSocketTestSession
120      -      with test_client.websocket_connect(
121      -          "/api/events/in", subprotocols=["prefect"]
122      -      ) as websocket:
123      -          auth_message = {
124      -              "type": "auth",
125      -              "token": "my-token",
126      -          }
127      -          websocket.send_json(auth_message)
128      -          message = websocket.receive_json()
129      -          assert message["type"] == "auth_success"
130      -
         125    +     # When auth is not configured, no subprotocol or auth handshake is required
         126    +     with test_client.websocket_connect("/api/events/in") as websocket:
131      127            websocket.send_text(event1.model_dump_json())
132      128            websocket.send_text(event2.model_dump_json())
133      129
                ↓
```

Comments 0

Please sign in to comment.