

Eugene Agent Auth — End-to-End Flow

Developer walkthrough: UI MS Entra access token → HTTPBearer dependency → PyJWT HS256 validation → allowlist check → downstream bearer forwarding to eugene-mcp

Audience

Engineers working on the **eugene-agent-ws** FastAPI service (the Strands ReAct agent behind the Next.js chat UI). This guide walks the auth surface exposed at `/agent/api/auth/whoami` plus the two reusable dependencies — `get_current_user` and `get_current_token` — that every protected query route declares. Each reference uses the form `path/to/file.py:line`.

Two auth surfaces, one service

Eugene has two FastAPI services and they do *not* share an auth router. The **main service** `src/foundation/router/auth_router.py` runs the MS Entra ID Authorization Code login + JWT mint (documented separately). The **agent-ws** service covered here is a *pure resource server*: it never mints tokens. It accepts a bearer token already issued by the main service (or directly by Entra in some dev setups), validates it locally with PyJWT, and propagates the same raw token downstream to the MCP server when the agent invokes tools.

Caveat

The router only exposes one endpoint (GET `/auth/whoami`); the heavy lifting is in the FastAPI dependencies that every query route shares. Read sections 2 and 4 together to see how a single inbound token authorises the request and then propagates into the MCP HTTP client.

0. Auth model (read this first)

The agent-ws service treats every inbound HTTP request as authenticated *only* if it carries a valid Eugene access token in an Authorization: Bearer <jwt> header. The expected token shape is fixed by router/auth/const.py:

```
EUGENE_TENANT_ID      = eugene_tenant_id()
EUGENE_CLIENT_ID      = eugene_client_id()
EUGENE_CLIENT_SECRET  = eugene_client_secret()
EUGENE_TOKEN_ISSUER   = f"https://eugene.ai.cs1g1.cs1g.net/{EUGENE_TENANT_ID}"
EUGENE_TOKEN_ALGORITHM = "HS256"
EUGENE_TOKEN_AUDIENCE = f"api://eugene/{EUGENE_CLIENT_ID}"
EUGENE_AGENT_ALLOWLIST = eugene_agent_allowlist()
```

- **Issuer (iss):** https://eugene.ai.cs1g1.cs1g.net/<tenant> — the main Eugene service when it mints a token after the Entra OIDC exchange.
- **Audience (aud):** api://eugene/<client_id> — both **eugene-ws** and **eugene-agent-ws** accept the same audience because they share EUGENE_CLIENT_ID.
- **Algorithm:** HS256 (symmetric, signed with EUGENE_CLIENT_SECRET). The agent-ws process must be started with the same secret as the issuing main service.
- **Required claims:** exp, iat, aud, iss, tid, sub, roles, upn. The first four are checked by PyJWT options; the last four are checked explicitly in validate_eugene_access_token.
- **Allowlist:** EUGENE_AGENT_ALLOWLIST is a pipe-delimited set of upn values. An empty set means 'allow any authenticated user' (dev default); a populated set acts as a hard ACL on top of the JWT check.

Note on Entra ID

The UI (**eugene-agent-ui-next**) acquires a Microsoft Entra ID access token via MSAL and posts it as the bearer. The agent service does *not* call Microsoft directly — it trusts the symmetric signature, which is valid because the main Eugene service re-mints an HS256 token in front of the raw Entra token. The plumbing on the UI side is agents/eugene-agent-ui-next/app/api/auth/token/route.ts; this guide treats it as a black box producing a Eugene JWT.

1. HTTP entry — the router

`agents/eugene-agent-ws/src/router/auth/auth_router.py`

The router itself is intentionally tiny — nine lines of logic. Only one endpoint is exposed; everything protected lives under other routers (query, health, root) that reuse the same dependencies.

```
router = APIRouter(
    prefix="",
    tags=["auth"],
)

@router.get("/auth/whoami")
def protected(user: dict = Depends(get_current_user)):
    return {"user": user}
```

- Mount point: `eugene_chat_ws.py:30-37` calls `add_routers(app)` which imports `router.auth.auth_router` and registers it.
- FastAPI app declares `root_path="/agent/api"` in `eugene_chat_ws.py:53-57`, so the deployed URL is `GET /agent/api/auth/whoami` (behind ALB + Cognito on AWS, plain on `localhost:18001`).
- Rate limiting via `slowapi` is applied app-wide (`eugene_chat_ws.py:60-63`) — the auth endpoint inherits the default limiter.
- No path or query params. The single 'input' is the `Authorization` header consumed by `HTTPBearer`.
- Returns `{"user": <claims dict>}` on 200; 401 on missing/invalid token; 403 on allowlist denial.

Endpoint reference

```
GET /agent/api/auth/whoami
Headers : Authorization: Bearer <eugene-jwt>
200      : { "user": { "tid": ..., "sub": ..., "upn": ..., "roles": [...] } }
401      : { "detail": "Invalid or expired token" }
403      : { "detail": "Access denied" }
```

2. Token validation internals

[agents/eugene-agent-ws/src/router/auth/auth.py](#)

Two dependencies live here. Every protected route picks one or both. They both source the bearer credential from `SECURITY = HTTPBearer(...)` defined in `router/auth/conf.py:6`.

`get_current_user (auth.py:13)`

```
def get_current_user(
    credentials: HTTPAuthorizationCredentials = Depends(SECURITY),
):
    try:
        token = credentials.credentials
        user = validate_eugene_access_token(token)
    except HTTPException:
        raise
    except Exception as exc:
        # SEC-02 fix: invalid tokens must surface as 401, not 500
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid or expired token",
            headers={"WWW-Authenticate": "Bearer"},
        ) from exc

    user_name = user.get("upn", "") if isinstance(user, dict) else ""
    if not EUGENE_AGENT_ALLOWLIST or user_name in EUGENE_AGENT_ALLOWLIST:
        return user

    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN,
        detail="Access denied",
    )
```

- Order of operations: (a) extract bearer string, (b) validate via PyJWT, (c) normalise exceptions to a 401, (d) apply allowlist → 403 if denied.
- The bare `except HTTPException: raise` preserves any 401/403 raised inside `validate_eugene_access_token` (e.g. "Invalid token structure").
- The SEC-02 fix is load-bearing: before it, a malformed JWT would bubble out as a 500. The catch-all rewrites anything non-`HTTPException` as a 401 with the proper `WWW-Authenticate: Bearer` header.

`get_current_token (auth.py:43)`

```
async def get_current_token(
    credentials: HTTPAuthorizationCredentials = Depends(SECURITY),
):
    return credentials.credentials
```

Pure pass-through: returns the raw bearer string. This is the dependency used by `/query` and `/query/stream` in `query/router/chat_query_agent_router.py:38, 86` so the same token can be forwarded to MCP (see section 4). It does **not** re-validate — routes that need both validation *and* raw forwarding declare both deps:

```
token: str = Depends(get_current_token)
user: dict = Depends(get_current_user)
```

2b. PyJWT decode — the actual signature check

`agents/eugene-agent-ws/src/router/auth/eugene_jwt.py`

JWT library: **PyJWT** (import `jwt` at line 4). No JWKS, no async, no caching — HS256 symmetric verify only.

```
def validate_eugene_access_token(token: str) -> dict:
    if not token:
        raise HTTPException(status_code=401, detail="Authentication required")

    try:
        claims = jwt.decode(
            jwt=token,
            key=EUGENE_CLIENT_SECRET,
            algorithms=[EUGENE_TOKEN_ALGORITHM], # ["HS256"]
            audience=EUGENE_TOKEN_AUDIENCE,
            issuer=EUGENE_TOKEN_ISSUER,
            options={
                "require_exp": True,
                "require_iat": True,
                "verify_aud": True,
                "verify_iss": True,
            },
        )

        required_claims = ["tid", "sub", "roles", "upn"]
        missing_claims = [c for c in required_claims if c not in claims]
        if missing_claims:
            raise HTTPException(status_code=401, detail="Invalid token structure")

        return claims
    except Exception as e:
        raise HTTPException(status_code=401, detail="Authentication error")
```

- **Signature:** HS256 with the shared `EUGENE_CLIENT_SECRET`. Any drift between the main service's secret and the agent service's secret manifests as a 401 here.
- **Time claims:** `exp` and `iat` are required (PyJWT raises `ExpiredSignatureError` on expired `exp`). `nbft` is not enforced.
- **iss/aud:** enforced by PyJWT options; mismatch raises `InvalidIssuerError` or `InvalidAudienceError` → caught and surfaced as 401.
- **Custom claims:** `tid`, `sub`, `roles`, `upn` checked explicitly. Missing any one of them → 401 with "Invalid token structure".
- **Broad except:** the final `except Exception` is intentionally broad to keep error details out of responses; the only client-visible message is "Authentication error".

3. Response model — user payload shape

There is no pydantic `response_model` on the auth route — the handler returns a raw dict. The shape is whatever PyJWT decoded, wrapped in a "user" key:

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "user": {
    "iss": "https://eugene.ai.cslg1.cslg.net/<tenant-id>",
    "aud": "api://eugene/<client-id>",
    "iat": 1737000000,
    "exp": 1737003600,
    "tid": "<entra-tenant-guid>",
    "sub": "<entra-object-id>",
    "upn": "jane.doe@example.com",
    "roles": ["eugene.agent.user"]
  }
}
```

- upn is what the chat router logs (`chat_query_agent_router.py:42`, `chat_query_agent_router.py:90`) and what the allowlist filter compares against.
- roles is currently informational — no RBAC decisions are made on it inside agent-ws. The allowlist is the only authorisation gate.
- Because there is no `response_model`, any extra claims your IdP includes (e.g. email, name) will pass through verbatim. Treat the response as informational, not as a stable contract.

Failure shapes

401 (no header / bad signature / expired / bad iss/aud / missing claim)
{ "detail": "Invalid or expired token" }
WWW-Authenticate: Bearer

401 (claims missing tid/sub/roles/upn) - upstream message
{ "detail": "Invalid token structure" }

403 (signed token but upn not in EUGENE_AGENT_ALLOWLIST)
{ "detail": "Access denied" }

4. Token forwarding to eugene-mcp

Every protected query route declares `token: str = Depends(get_current_token)` and passes that token down into the Strands agent so the MCP HTTP client can re-attach it on every tool call. This is the 'trusted subsystem with on-behalf-of forwarding' pattern — agent-ws does not re-mint, swap, or downgrade the token; the same JWT that authorised the inbound HTTP request authorises every outbound MCP call it triggers.

Route → agent (chat_query_agent_router.py:51-56)

```
response = eugene_agent.execute(
    token=token,                    # raw bearer from get_current_token
    user_prompt=prompt,
    conversation_id=chat_request.conversation_id,
    include_tools=tools,
)
```

Agent → MCP client (eugene_data_agent.py:283-295)

```
def _build_mcp_client(self, token: str) -> MCPClient:
    # SEC-01 fix: default verify=True; dev can opt out via
    # EUGENE_MCP_VERIFY_SSL=false
    verify = _MCP_VERIFY_SSL
    return MCPClient(
        lambda: streamable_http_client(
            url=self.eugene_mcp_server_url,
            http_client=httpx.AsyncClient(
                verify=verify,
                headers={"Authorization": f"Bearer {token}"},
                timeout=httpx.Timeout(
                    connect=10.0, read=60.0, write=30.0, pool=10.0
                ),
            ),
        ),
    )
```

- The header is built once per request: each chat invocation constructs a fresh `httpx.AsyncClient` with the current user's bearer baked into its default headers.
- MCP itself validates the same JWT independently — see the MCP auth router. A token that passes agent-ws but fails MCP (e.g. secret rotation skew between services) surfaces as a 401 from the MCP tool call, which Strands wraps as a tool error.
- SEC-01: TLS verify defaults to true. Override only in dev with `EUGENE_MCP_VERIFY_SSL=false`.
- There is **no** token caching, no refresh, and no Entra OBO exchange in agent-ws — the inbound and outbound tokens are byte-identical for the lifetime of the chat call.
- If the chat call exceeds the JWT's exp, subsequent MCP tool calls inside that ReAct loop will start 401'ing. There is no automatic retry.

5. Suggested debugging walk

- Bring the stack up locally: `docker-compose up eugene_neo4j eugene_ws eugene_mcp` then run the agent-ws on the host with `python -m uvicorn eugene_chat_ws:app --reload --port 18001`.
- Mint a token by signing in through the Next.js UI and grabbing the bearer from the network tab (header on any `/agent/api/...` request) — or call the main service's `POST /auth/token` directly.
- Smoke test: `curl -H "Authorization: Bearer $TOKEN" http://localhost:18001/agent/api/auth/whoami` → expect 200 with the claims dict. Strip the header to confirm 401 with `WWW-Authenticate: Bearer`.
- Breakpoint `router/auth/eugene_jwt.py:22` (the `jwt.decode` call) to inspect raw claims before the required-claims check. PyJWT will already have validated `exp/iat/aud/iss` by the time you land on the next line.
- Breakpoint `router/auth/auth.py:32` to watch the allowlist branch — if a user gets a surprise 403, this is almost always a stale `EUGENE_AGENT_ALLOWLIST` env var.
- For MCP forwarding, breakpoint `query/agent/eugene_data_agent.py:291` and confirm the bearer string matches the one you sent. Mismatches usually mean a dependency override is constructing the agent at module load time instead of per-request.

6. Reference index

Auth model / config

<code>agents/eugene-agent-ws/src/router/auth/conf.py</code>	<code># HTTPBearer, env getters</code>
<code>agents/eugene-agent-ws/src/router/auth/const.py</code>	<code># iss/aud/alg/allowlist</code>

HTTP entry

<code>agents/eugene-agent-ws/src/router/auth/auth_router.py</code>	<code># GET /auth/whoami</code>
<code>agents/eugene-agent-ws/src/eugene_chat_ws.py</code>	<code># add_routers, root_path=/agent/api</code>

Dependencies + validation

<code>agents/eugene-agent-ws/src/router/auth/auth.py</code>	<code># get_current_user, get_current_token</code>
<code>agents/eugene-agent-ws/src/router/auth/eugene_jwt.py</code>	<code># PyJWT decode + required-claims check</code>

Token forwarding

<code>agents/eugene-agent-ws/src/query/router/chat_query_agent_router.py</code>	<code># token=Depends(get_current_token)</code>
<code>agents/eugene-agent-ws/src/query/agent/eugene_data_agent.py:283-295</code>	<code># MCPClient bearer header</code>

UI side (out of scope for this guide)

<code>agents/eugene-agent-ui-next/app/api/auth/token/route.ts</code>	<code># MSAL token acquisition</code>
--	---------------------------------------