New issue

Reflected Cross-Site Scripting (XSS) via Task Name in gpt-researcher #1692

Open

August829 opened 2 weeks ago

Product Information

- **Vendor:** assafelovic
- **Product:** GPT Researcher
- **Affected Version:** <= 3.4.3
- **Fixed Version:** N/A (unpatched)
- **Repository:** https://github.com/assafelovic/gpt-researcher
- **Component:** `backend/server/server_utils.py`, `frontend/scripts.js`
- **Discoverer:** Yu Bao
- **Date:** 2026-03-19

Vulnerability Summary

GPT Researcher v3.4.3 and earlier versions are vulnerable to Reflected Cross-Site Scripting (XSS) via the research task name. When a user submits a research query containing HTML/JavaScript through the WebSocket interface, the backend includes the unsanitized task name in multiple WebSocket `logs` response messages. The lightweight frontend renders these log messages using `innerHTML` without any sanitization, causing the injected script to execute in the user's browser. No authentication is required to trigger this vulnerability.

Vulnerability Details

CWE: CWE-79 (Improper Neutralization of Input During Web Page Generation)
CVSS v3.1 Score: 6.1 (Medium)
CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

Root Cause

The backend embeds the user-supplied `task` string directly into WebSocket log message outputs without HTML encoding. The lightweight frontend receives these messages and renders them via `innerHTML`, creating a reflected XSS vulnerability.

Vulnerable Code — Backend (Log Message Generation)

The backend includes the raw task name in multiple log messages throughout the research pipeline:

`gpt_researcher/actions/utils.py` and `gpt_researcher/skills/researcher.py` — Log messages containing raw task:

```
# Multiple locations generate log messages like:
await stream_output("logs", "starting_research",
    f"🔍 Starting the research task for '{task}'...", websocket)

await stream_output("logs", "planning_research",
    f"🌐 Browsing the web to learn more about the task: {task}...", websocket)
```

The `task` variable contains the user's raw input with no HTML encoding applied.

Vulnerable Code — Frontend (Unsafe Rendering)

`frontend/scripts.js:1042-1046` — `addAgentResponse()`:

```
const addAgentResponse = (data) => {
    const output = document.getElementById('output');
    const responseDiv = document.createElement('div');
    responseDiv.className = 'agent_response';
    responseDiv.innerHTML = data.output; // ← XSS SINK: raw HTML from server rendered direct
    output.appendChild(responseDiv);
};
```

`frontend/scripts.js:870-874` — WebSocket message handler:

```
if (data.type === 'logs') {
    if (data.content === 'subqueries' && data.metadata && Array.isArray(data.metadata)) {
        displaySubQuestions(data.metadata)
    }
```

```
    addAgentResponse(data)  // Every logs message rendered via innerHTML
}
```

Complete Data Flow

```
1. User enters task: <img src=x onerror=alert(document.domain)>

2. WebSocket sends:
   start {"task":"<img src=x onerror=alert(document.domain)>", ...}

3. Backend generates log messages containing raw task name:
   {"type":"logs","output":"🔍 Starting the research task for '<img src=x
onerror=alert(document.domain)>'..."}

4. Frontend receives WebSocket message:
   scripts.js:862  → data = JSON.parse(event.data)
   scripts.js:870  → if (data.type === 'logs')
   scripts.js:874  → addAgentResponse(data)

5. addAgentResponse renders via innerHTML:
   scripts.js:1046 → responseDiv.innerHTML = data.output

6. Browser parses <img>, src=x fails, onerror fires → alert() executes
```

Impact

1. **Self-XSS to Social Engineering:** An attacker can instruct a victim to paste a specific "research query" containing a JavaScript payload. The XSS executes immediately when the research starts.
2. **Session Hijacking:** The injected JavaScript runs in the context of the application's origin, with access to cookies, localStorage, and the DOM.
3. **Multiple Injection Points:** A single malicious task name triggers XSS through at least 4 separate log messages during one research session, each executing independently:
   - `starting_research`: "Starting the research task for '...'"
   - `mcp_retrieval_stage1`: "Selecting optimal MCP tools for: ..."
   - `mcp_no_results`: "No relevant information found via MCP for: ..."
   - `planning_research`: "Browsing the web to learn more about the task: ..."

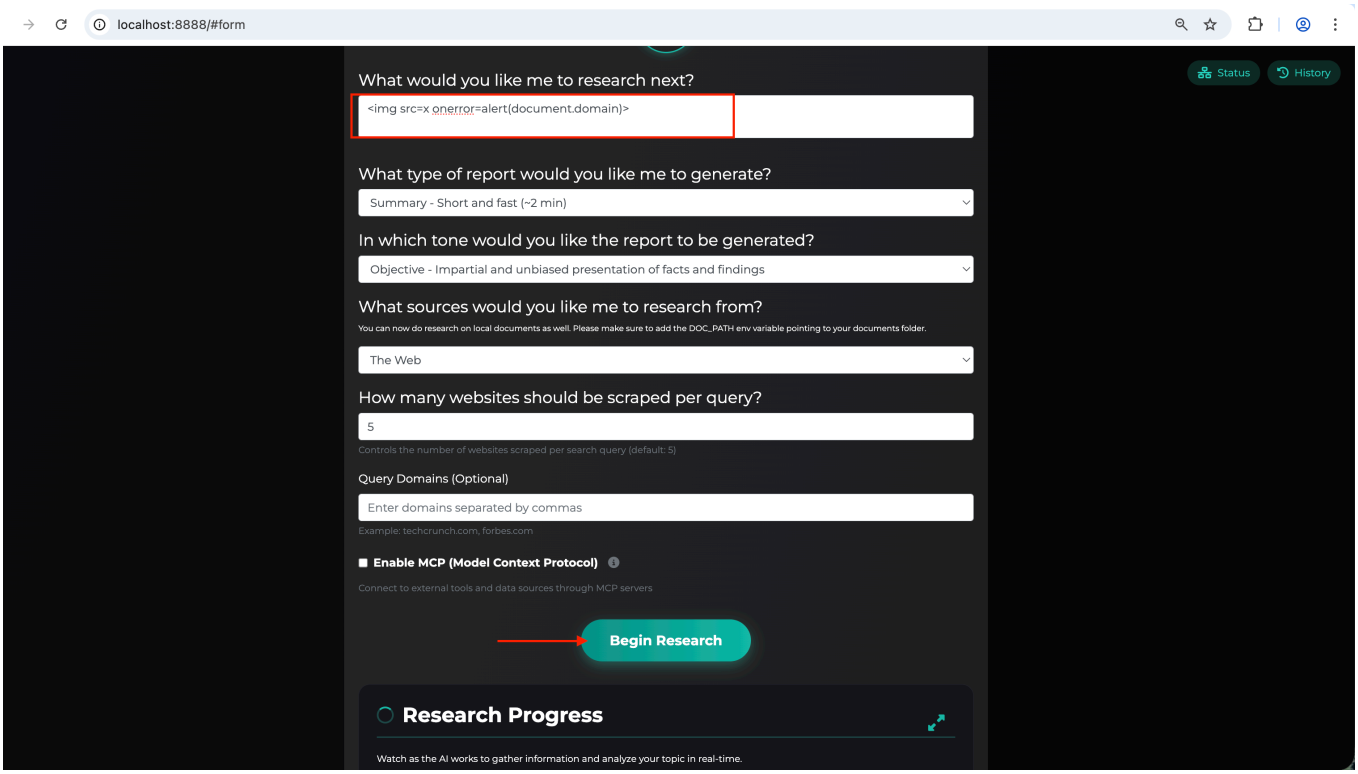
Proof of Concept

Prerequisites

A running instance of gpt-researcher v3.4.3 with the lightweight frontend at `http://localhost:8888`.

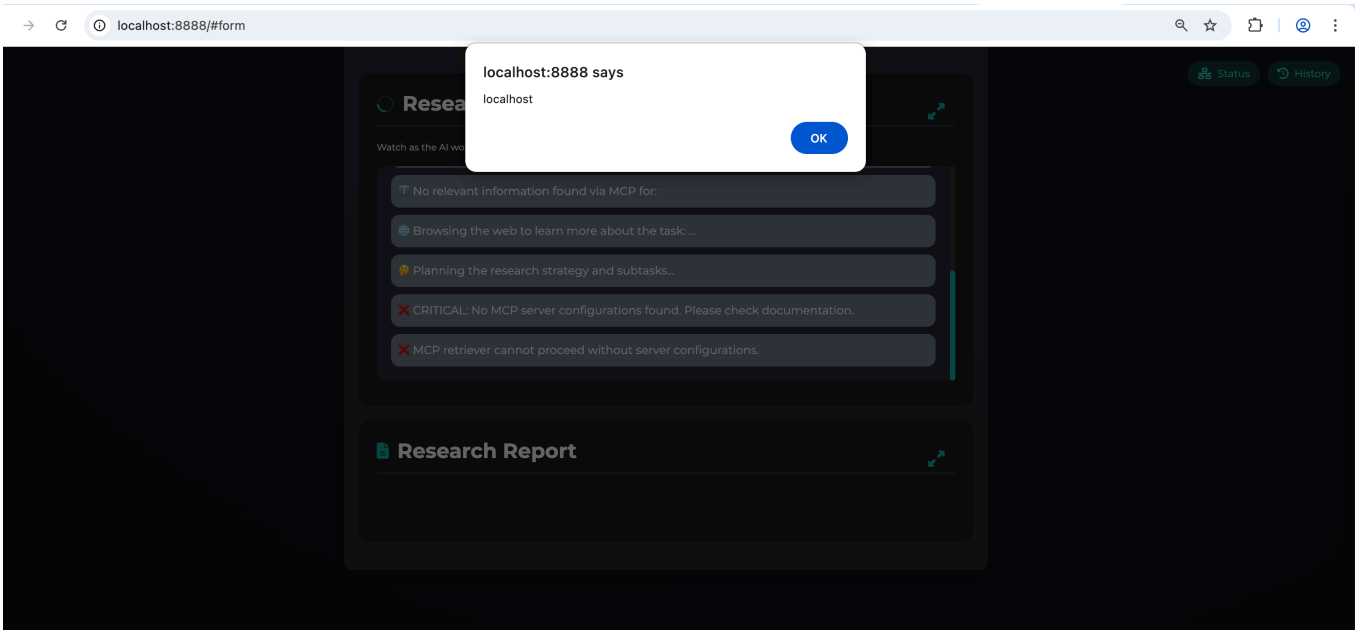
Method 1: Direct Browser Input

Open `http://localhost:8888`, enter the following in the search box, and click "Research":




```
<img src=x onerror=alert(document.domain)>
```

The XSS fires immediately as the first log message is rendered.



Verified Results

Tested against a live gpt-researcher v3.4.3 instance on `localhost:8888`.

WebSocket responses containing unsanitized HTML (verbatim from server):

```
[XSS IN LOGS] content=starting_research
  output: 🔍 Starting the research task for '<img src=x
onerror=alert(document.domain)>'...

[XSS IN LOGS] content=mcp_retrieval_stage1
  output: 🧠 Stage 1: Selecting optimal MCP tools for: <img src=x
onerror=alert(document.domain)>

[XSS IN LOGS] content=mcp_no_results
  output: ℹ️ No relevant information found via MCP for: <img src=x
onerror=alert(document.domain)>

[XSS IN LOGS] content=planning_research
  output: 🌐 Browsing the web to learn more about the task: <img src=x
onerror=alert(document.domain)>...
```

All 4 messages flow to `addAgentResponse()` → `responseDiv.innerHTML = data.output` → XSS executes.

Browser alert popup confirmed on `localhost:8888` with `alert(document.domain)` showing "localhost".

Remediation

Fix: HTML-Escape Log Output Before innerHTML

`frontend/scripts.js` — Sanitize in `addAgentResponse()`:

```
const addAgentResponse = (data) => {
    const output = document.getElementById('output');
    const responseDiv = document.createElement('div');
    responseDiv.className = 'agent_response';
    responseDiv.textContent = data.output; // Use textContent instead of innerHTML
    output.appendChild(responseDiv);
};
```

Or if HTML formatting is needed in log messages:

```
const addAgentResponse = (data) => {
    const output = document.getElementById('output');
    const responseDiv = document.createElement('div');
    responseDiv.className = 'agent_response';
    responseDiv.innerHTML = DOMPurify.sanitize(data.output); // Sanitize with DOMPurify
```

```
    output.appendChild(responseDiv);
};
```

Backend Fix: HTML-encode task name in log messages

```
import html

await stream_output("logs", "starting_research",
    f"🔍 Starting the research task for '{html.escape(task)}'...", websocket)
```

References

- CWE-79: https://cwe.mitre.org/data/definitions/79.html
- OWASP XSS Prevention: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Scripting_Prevention_Cheat_Sheet.html

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