Command Injection in MCP Server foundry-mcp-server

Package

foundry-mcp-server (GitHub)

@pranesh.asp/foundry-mcp-server (npm)

Affected versions

<= 0.1.5

Patched versions

None

Description

The MCP Server at https://github.com/PraneshASP/foundry-mcp-server/ 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 13 tools that rely on Node.js child process API exec () 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:

- ***cast_call***: vulnerable args are *contractAddress, functionSignature, args, rpcUrl, blockNumber, from*
- ***cast_send***: vulnerable args are *contractAddress, functionSignature, args, from, value, rpcUrl, gasLimit, gasPrice, confirmations*
- ***cast_balance***: vulnerable args are *address, rpcUrl, blockNumber*
- ***cast_receipt***: vulnerable args are *txHash, rpcUrl, confirmations, field*
- ***cast_chain***: vulnerable arg is *rpcUrl*
- ***anvil_start***: vulnerable args are *port, blockTime, forkUrl, forkBlockNumber, accounts, mnemonic*
- ***forge_script***: vulnerable args are *scriptPath, sig, rpcUrl*
- ***convert_eth_units***: vulnerable arg is *value*
- ***heimdall_disassemble***: vulnerable args are *target, rpcUrl, fileName, outputDir*
- ***heimdall_decode***: vulnerable args are *target, rpcUrl, openaiApiKey*
- ***heimdall_decompile***: vulnerable args are *target, rpcUrl, fileName, timeout, outputDir*
- ***heimdall_cfg***: vulnerable args are *target, rpcUrl, fileName, timeout, outputDir*
- ***heimdall_inspect***: vulnerable args are *target, rpcUrl, transposeApiKey, fileName, outputDir*

All these tools construct Foundry CLI commands by concatenating user-controlled parameters directly into shell commands executed via execAsync().

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

cast_call:

call chain:

1. MCP client calls the ***cast_call*** tool with attacker-controlled parameters such as *contractAddress*, *functionSignature*, *args*, *rpcUrl*, *blockNumber*, and *from*
2. Tool handler in src/tools/cast/call.ts receives these parameters without strict validation or sanitization

3. The handler constructs a shell command string by concatenating these attacker-controlled parameters
4. The constructed command string is passed to executeCommand() in src/utils/command.ts
5. executeCommand() invokes execAsync() in src/utils/command.ts
6. execAsync() is a Promise wrapper around child_process.exec(), allowing shell metacharacters to be interpreted and resulting in command injection

Vulnerable lines of code:

(*Parameter definition without validation*): https://github.com/PraneshASP/foundry-mcp-server/blob/main/src/tools/cast/call.ts#L10-L17

(*Command construction with user input*): https://github.com/PraneshASP/foundry-mcp-server/blob/main/src/tools/cast/call.ts#L28-L46

(*Unsafe execAsync() call*): https://github.com/PraneshASP/foundry-mcp-server/blob/main/src/utils/command.ts#L55-L57

```
55  export async function executeCommand(command: string): Promise<{success: boolean, message: string}> {
56    try {
57      const { stdout, stderr } = await execAsync(command);
```

[**The remaining 12 tools follow the similar calling pattern**]

PoC: Using Inspector

[Note: we only test the ***cast_call*** tool, the same procedure applies to ***all other tools***]

[Environment: Windows 10 + WSL2]

1. *Install Foundry toolchain*

- install WSL2
- in wsl, install Foundry toolchain (curl -L https://foundry.paradigm.xyz | bash && foundryup)

2. *Start the MCP server:*

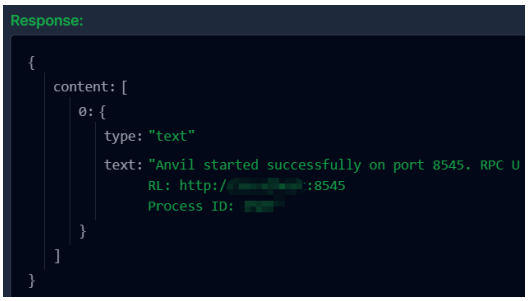
- in wsl, clone this server
- bun i && bun build ./src/index.ts --outdir ./dist --target node

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 ***anvil_start*** tool
- click Run Tool
- the response is shown below:



(This is mainly to start a local Ethereum blockchain node to provide a local blockchain environment for smart contract development and testing)

- select the ***anvil_status*** tool
- click Run Tool
- the response is shown below:

(This is mainly to verify whether the local development blockchain node is operational before attempting to interact with it through other tools like ***cast_call***)

5. Verify the file poc.txt does not exist:

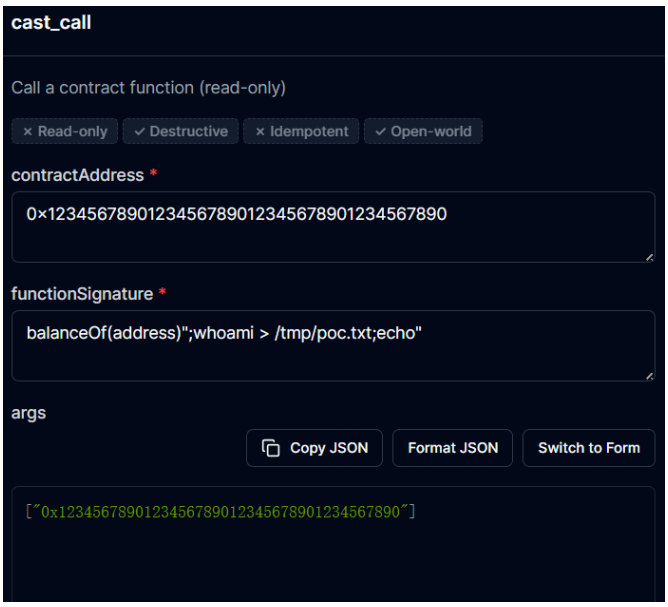
- ls /tmp/poc.txt
- No such file or directory

6. select the ***cast_call*** tool

in the ***functionSignature*** field, input:

balanceOf(address)";whoami > /tmp/poc.txt;echo"

(In the *contractAddress* field, input: 0x1234567890123456789012345678901234567890. In the *args* field, input: ["0x1234567890123456789012345678901234567890"]. And all other parameters use their default values)



7. Click Run Tool

the command executed will be:

/root/.foundry/bin/cast call 0x1234567890123456789012345678901234567890 "balanceOf(addr

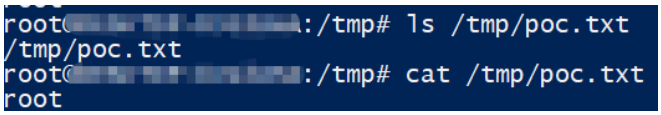
ess)";whoami > /tmp/poc.txt;echo"" 0x1234567890123456789012345678901234567890

8. *Observe the request being sent:*

9. Response:

10. *Confirm that the injected command executed:*

- cat /tmp/poc.txt
- root



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 exec. Use execFile 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 parameters mentioned above.

- 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

Credit

Discovered and reported by Yinci Chen.