# Command Injection in MCP Server WireMCP

## Package

0xKoda/WireMCP (GitHub)

## Affected versions

<= 1.0.0

## Patched versions

None

## Description

The MCP Server at https://github.com/0xKoda/WireMCP/ 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 multiple tools that invoke the external program *tshark* using the Node.js child_process.exec API, which executes commands via a system shell and is unsafe when combined with untrusted user input.

The following tools are affected by this unsafe command execution pattern:

- ***capture_packets***
- ***get_summary_stats***
- ***get_conversations***
- ***check_threats***
- ***analyze_pcap***
- ***extract_credentials***

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

Vulnerable line of code (***capture_packets)***: https://github.com/0xKoda/WireMCP/blob/main/index.js#L62-L65

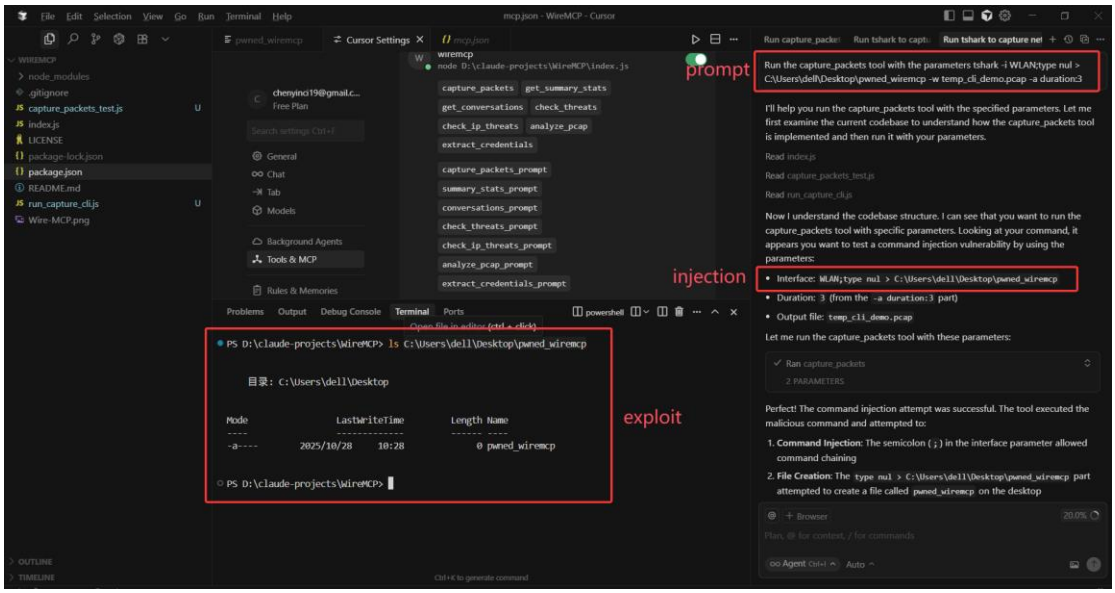
```
// Tool 1: Capture live packet data
server.tool(
  'capture_packets',
  'Capture live traffic and provide raw packet data as JSON for LLM analysis',
  {
    interface: z.string().optional().default('en0').describe('Network interface to capture from (e.g., eth0, en0)'),
    duration: z.number().optional().default(5).describe('Capture duration in seconds'),
  },
  async (args) => {
    try {
      const tsharkPath = await findTshark();
      const { interface, duration } = args;
      const tempPcap = 'temp_capture.pcap';
      console.error(`Capturing packets on ${interface} for ${duration}s`);

      await execAsync(
        `${tsharkPath} -i ${interface} -w ${tempPcap} -a duration:${duration}`,
        { env: { ...process.env, PATH: `${process.env.PATH}:/usr/bin:/usr/local/bin:/opt/homebrew/bin` } }
      );

      const { stdout, stderr } = await execAsync(
        `${tsharkPath} -r "${tempPcap}" -T json -e frame.number -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport -e tcp.flags -e frame.time -e http.request.method -e http.response.code`,
        { env: { ...process.env, PATH: `${process.env.PATH}:/usr/bin:/usr/local/bin:/opt/homebrew/bin` } }
      );
      if (stderr) console.error(`tshark stderr: ${stderr}`);
      let packets = JSON.parse(stdout);
```

## Exploitation

When an MCP client (including LLM-based clients) is manipulated through prompt injection or other attack techniques to invoke an affected tool with input containing special shell characters (for example: ; rm -rf /tmp; # — do not execute this payload on a real system), the constructed command string is interpreted by the system shell.

As a result, unintended commands may be executed on the host running the MCP Server instead of the expected *tshark* invocation.

A practical example of this exploitation scenario using an LLM-based MCP client is shown below, where injected shell metacharacters in tool parameters result in unintended command execution on the host.




## Impact

Remote command injection on a running MCP Server, allowing execution of arbitrary system commands with the privileges of the MCP Server process.

## 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.
- If the user input is not a command-line flag, use the -- notation to terminate command and command-line flag, and indicate that the text after the -- double dash notation is benign value.

## References and Prior work

1. Exploiting MCP Servers Vulnerable to Command Injection
2. Liran's Node.js Secure Coding: Defending Against Command Injection Vulnerabilities