Vulnerability 1 - Timeout-based Driver Execution Delay (Tenant-Scoped DoS)

Summary

Although Homey bans some timeout APIs, apps/drivers can still insert **arbitrary delays** using standard JavaScript primitives (e.g., `Promise` + `setTimeout`). When such delays are compiled into a community app (especially a closed-source one), an attacker can introduce **significant latency** in safety-critical actions—e.g., smart locks, security systems, or cameras—resulting in a **tenant-scoped availability attack (DoS)**.

Details

- **Surface:** Capability handlers in device drivers (e.g., `onoff`) or higher-level app logic wrapping those capabilities.
- **Mechanism:** Wrap `setTimeout`/`this.homey.setTimeout` in a Promise and sleep before/after the critical action; JavaScript delays are stored as a **32-bit signed integer**, so the single-call maximum is ≈ **2,147,483,647 ms (24.85 days)**. Using **chunked sleeps** allows arbitrarily long delays.
- **Risk:** Closed-source apps can hide the delay in build artifacts; users only perceive “sluggish devices,” not a clear fault.

PoC

Minimal example (insert a 10-second delay in a capability handler):

```
if (value === false) {
  this.setClusterCapabilityValue('onoff', CLUSTER.ON_OFF, 0);
  const sleep = ms => new Promise(resolve => this.homey.setTimeout(resolve, ms));
  await sleep(10_000);                      // arbitrary pause
  this.setClusterCapabilityValue('onoff', CLUSTER.ON_OFF, 1);
}
```

Extended delay (bypass the 24.85-day cap via chunking):

```
const sleep = ms => new Promise(r => setTimeout(r, ms));
async function longSleep(msTotal) {
  const CHUNK = 2_147_483_647;             // ~24.85 days
  while (msTotal > 0) {
    await sleep(Math.min(CHUNK, msTotal));
    msTotal -= CHUNK;
  }
}
```

Impact

- **Type:** Tenant-scoped DoS / logic-bomb delay.
- **Effect:** Lock/unlock, arming, recording, or alert flows are delayed from **minutes to days**; repeated triggers can create periodic unavailability.
- **Scope:** The tenant that installs the malicious/buggy app; can extend to multiple devices within that tenant.

Mitigations

- **Platform constraints**
  - Prohibit or quota **blocking/long delays** (e.g., require declaration and UI warnings for ≥5 s);
  - Enforce **max handler execution time / watchdogs**; abort and alert on overruns;
  - For closed-source submissions, apply **bytecode/AST scanning** to detect large delays or chunked-sleep patterns.
- **Scheduling isolation**
  - Route delayed actions to **low-priority queues**; enforce **hard deadlines** for safety-critical capabilities (locks/alarms).
- **Observability**
  - Expose **per-app/device delay metrics**; alert on abnormal latency.

Vulnerability 2 - Memory Exhaustion via Buffer Allocations (Tenant-Scoped DoS)

Summary

An app can repeatedly call `Buffer.alloc()` (or similar) and **retain** the buffers to exhaust available memory, leading to **process/platform crash**. In testing, up to **~5 rapid calls** were sufficient; after the crash, **no device control was possible for ~2 minutes** with no notification, then the app/cloud lost connection and the hub's LED turned solid red until the user **unplugged to reboot**.

Details

- **Surface:** App code loops large allocations and keeps them in a global array to prevent GC.
- **Reference behavior:** Each call allocated **128 × 1 MiB** buffers (≈ **256 MiB** per call) and retained them; the practical kill threshold was **~1.3 GiB**, i.e., ≈5 calls.
- **Observed:** Short burst → OOM/crash; mobile app shows "Cannot connect...", ring LED solid red; manual power cycle required.

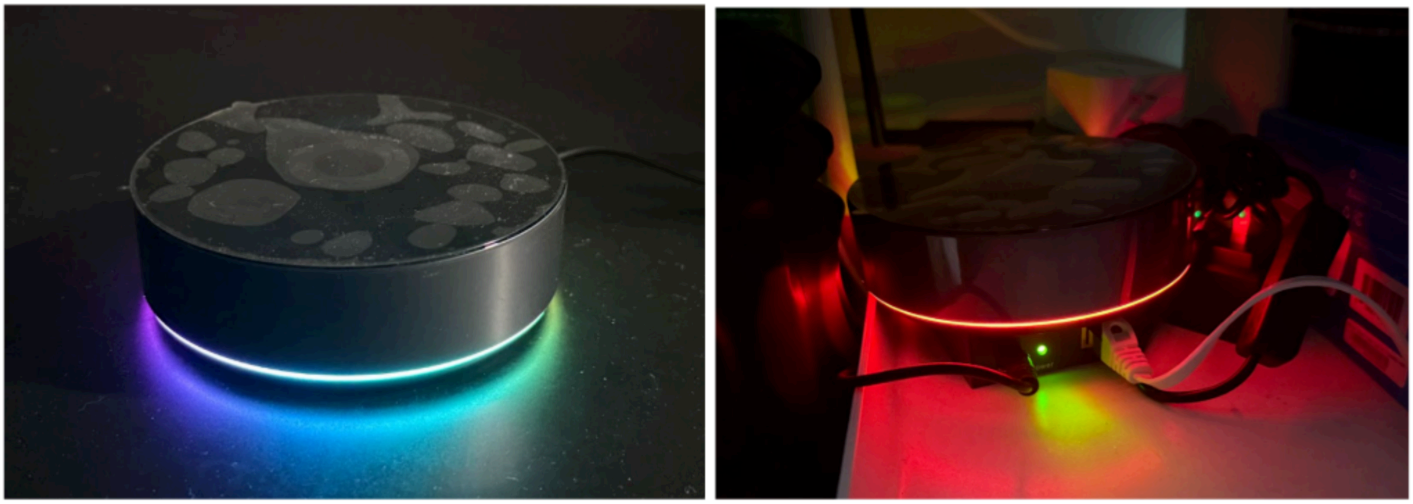
PoC

Minimal "leaker" module:

```
// memory-leak.js
module.exports.leakBuffers = async ({
  chunks = 128,            // buffers per call
  size   = 1024 * 1024,    // 1 MiB each
  fillByte = 0xAA,
  interval = 0,            // ms between allocations
} = {}) => {
  global.__buffers ??= [];
  for (let i = 0; i < chunks; i++) {
    global.__buffers.push(Buffer.alloc(size, fillByte));
    if (interval) await Homey.sleep(interval);
  }
};
```

Trigger (five quick rounds, typically enough to OOM):

```
for (let i = 0; i < 5; i++) {
  await leakBuffers({ chunks: 128, size: 1024 * 1024 });
}
```



(a) Normal (multi-color ring)

(b) Crashed (red ring)

Impact

- **Type:** Tenant-scoped DoS / resource exhaustion.
- **Effect:** Platform crash and **multi-minute outage**; requires **manual intervention**; repeatable for sustained unavailability.
- **Scope:** The tenant/hub running the offending app.

Mitigations

- **Resource isolation**

    - Impose **per-app memory hard limits** (e.g., cgroups/containers); on breach, **terminate and cool-down restart** the app without impacting others.
- **API safeguards**
    - Quota and rate-limit `Buffer.alloc`/`ArrayBuffer`; detect **global retention growth** patterns and deny.
- **Crash containment**
    - Ensure app crashes are **isolated** so the platform and other devices remain operable; surface **clear user alerts**.
- **Review & detection**
    - Apply **static scanning** (large allocations, global caches) and **dynamic sandbox runs** to flag OOM behavior before publishing.