

SocketIO / EngineIO DoS

May 2020

Quite a while ago, I reported an application Denial of Service vulnerability in the Socket.IO / Engine.IO parser implementations in nodejs and python. A single HTTP POST request can cause extreme CPU and memory usage, but in nodejs, a single HTTP POST request can even kill the server with a `Javascript heap out of memory` fatal error.

I assume some of what I've written is incorrect as I'm not an expert on v8 internals, but I do really love getting to the bottom of edge-case performance issues.

Protocol

The [engine.io protocol](#) allows bi-directional communication between a server and client, abstracting away the actual transport. The transport can be WebSockets, but if that isn't supported then another transport such as HTTP long polling is possible.

When the WebSocket transport is used, packets are encapsulated by the engine.io protocol. First there is a number specifying the [packet type](#). For instance, ping packets starting with `2` are sent as WebSocket data even though WebSocket has its own heartbeat mechanism. Sending the JSON data `{ "a": 123 }` requires prefixing with a `4`. The socket.io protocol on top of that, if used, will add a `2` prefix meaning [EVENT](#), so a WebSocket listener will receive `"42{\\"a\\":123}"`.

Using the long-polling transport, a payload containing multiple packets can be sent. In [version 3 of the protocol](#), the payload is encoded as:

```
<length1>:<packet1>[<length2>:<packet2>[...]]
```

e.g `6:42[{}]`11:4abcdefghij1:2 contains 3 packets:

1. Socket.io packet of length 6: Message (4), Event (2, `[{}]`)
2. Packet of length 11: Message (4), Data (`abcdefghij`)
3. Packet of length 1: Ping (2)

With WebSockets, the 3 packets would be sent separately. With HTTP long polling, the payload would be POSTed to `http(s)://host/socket.io/?EIO=3&transport=polling&sid=$SESSIONID`.

The denial-of-service bug lies in:

- Inefficient parsing of packets from payloads
- Maximum HTTP body size of 100MB

Make your 100MB count

You can send a payload containing 1e8 bytes to the server. That's quite a huge message, but how can we cause the server the most pain and suffering? The main methods are:

- **Many tiny packets:** send 25,000,000 empty event packets `2:422:422:422:42`
- **One giant int:** send the largest possible packet with integer data `99999991:42222222222222222222...`
- **Many heartbeats:** send 33,333,333 ping packets `1:21:21:21:2...`

Loading the body string into memory automatically eats up 100MB as a starting point, but it gets a hell of a lot worse.

Nodejs

With NodeJS, if the ping timeout (default 30s) is exceeded then the processing appears to be cancelled. Therefore, sending a payload which is so large it doesn't reach the memory exhausting step within the ping timeout will not kill the process. It will just waste CPU for 30 seconds. Sending a slightly smaller payload instead may cause the process to exit.

Many tiny packets

The bug here is due to [this 2016 changg](#). As the parser reads packets from the payload, it doesn't emit the `socket.onpacket` event immediately. Instead it queues up a new closure with `process.nextTick`. Since the next tick of the event loop doesn't come until all packets have been parsed, memory usage blows up.

All others

The python code seems to generally run slower than the nodejs code. Large payloads cause DoS primarily by wasting CPU time since python doesn't have a max heap size in the same way as v8. One giant int is slow as `int("2" * int(1e7))` is incredibly slow in python, perhaps because it allows Unicode digits like `𐀀` as well.

Exploit

I made a repo [bcaller/kill-engine-io](https://github.com/bcaller/kill-engine-io) containing test servers and code to trigger the DoS. Enjoy.

Servers with a lower max HTTP body size are less vulnerable. In fact, the default has been lowered in newer versions.

Tags: [security](#) | [poc](#) | [research](#) | [nodejs](#) | [python](#)

[\[blog by caller\]](#) Correspondence welcome at [blog \(@\) call.ee.8x9z](mailto:blog (@) call.ee.8x9z)