View online

Download PDF

The tale of avoiding a time-based DDOS attack in Node.js

Paolo Insogna

Node.js TSC, Principal Engineer @ **Platformatic**

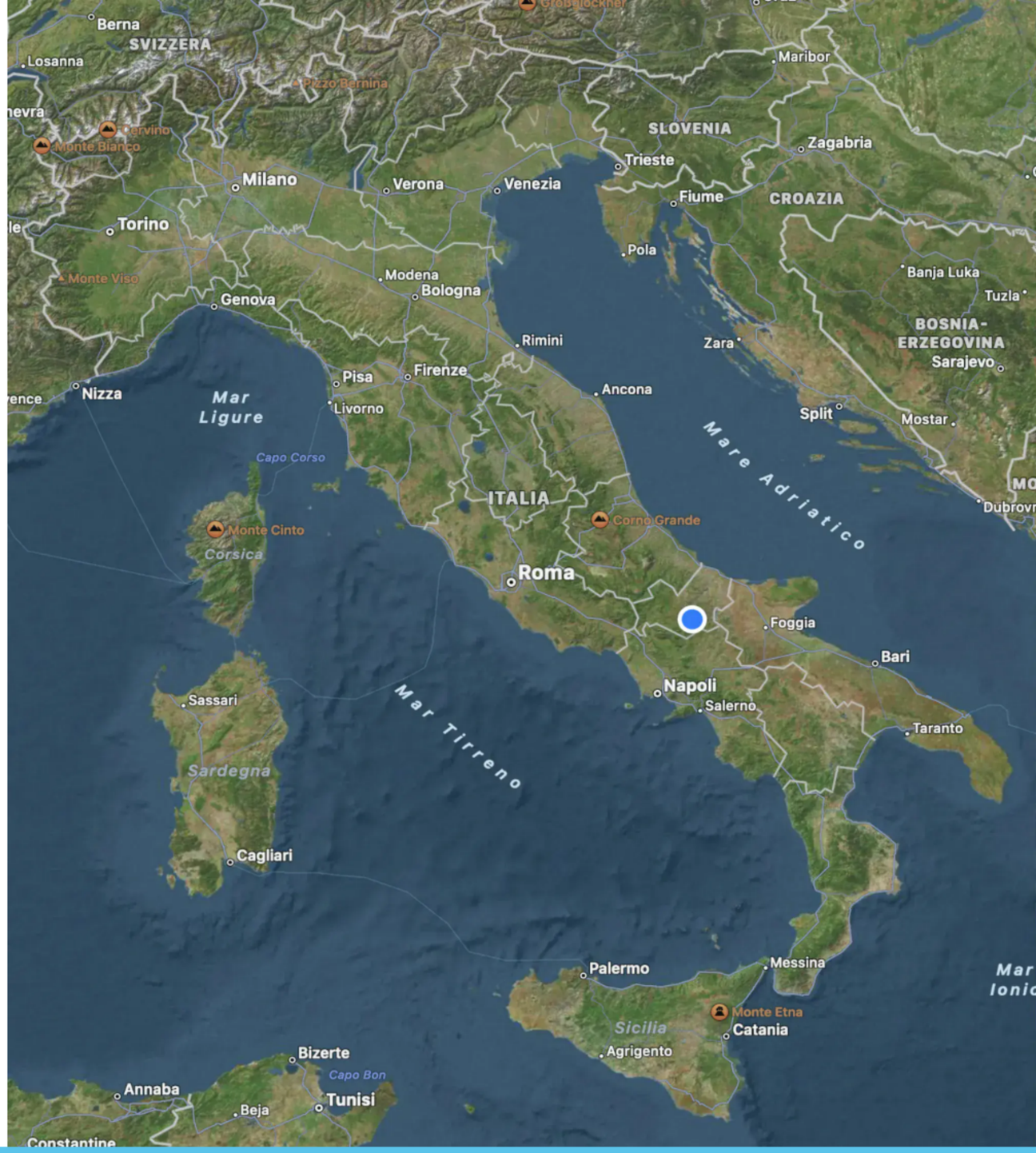
Sometimes, your worst enemy is slowness!

Hello, I'm **Paolo**!

Node.js Technical Steering Committee Member

Platformatic Principal Engineer

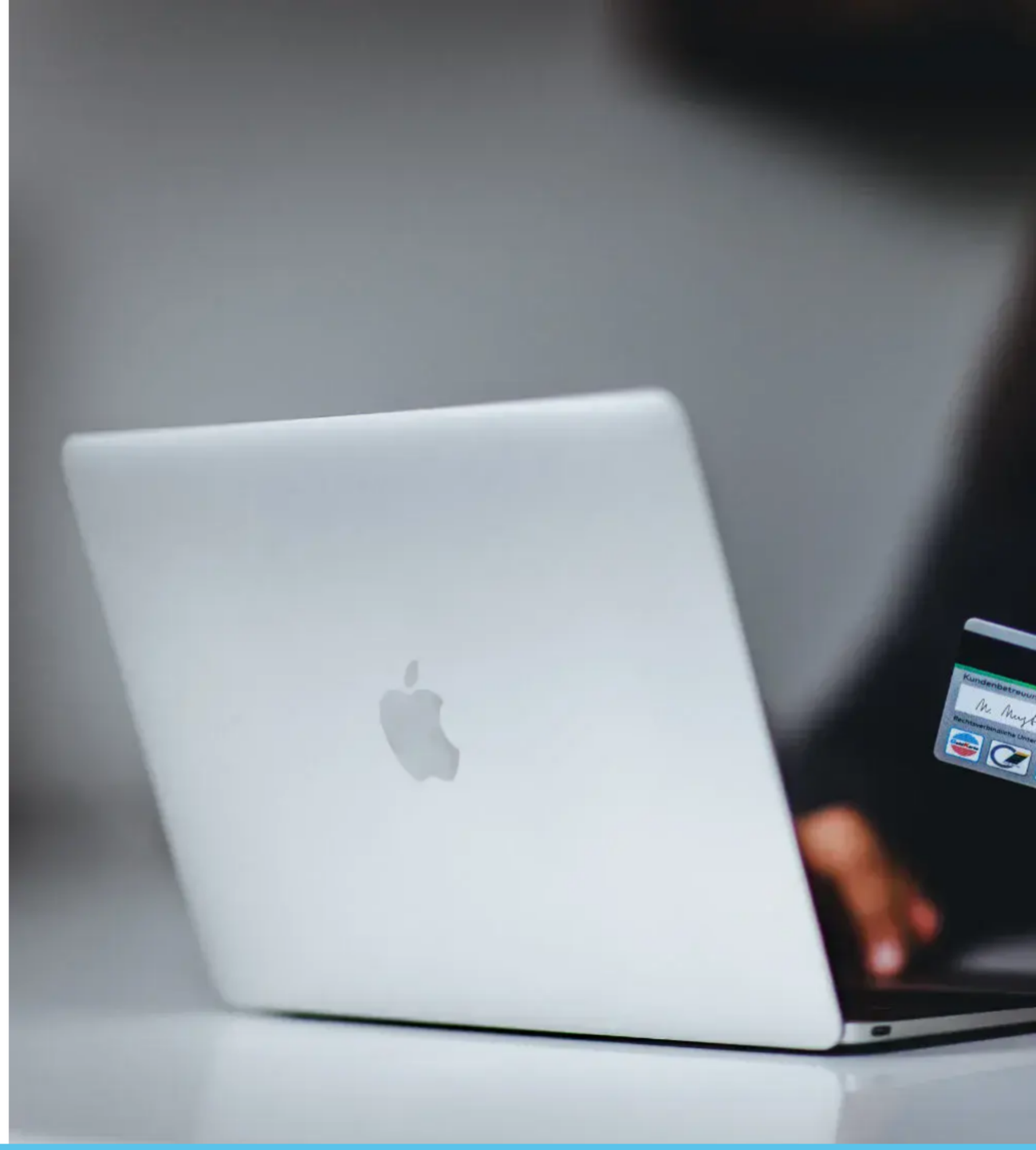
`paoloinsogna.dev` `ShogunPanda` `p_insogna` `pinsogna`



What do we use everyday?

Web applications play a very important part of our lives.

They must not go down!



We are all vulnerable!

Denial of Service Attack

Denial of Service Attack (DoS)
A network resource is maliciously made unavailable to its intended users.

Distributed Denial of Service (DDoS)
The incoming flooding traffic originates from many different sources.

In DDoS the attacker usually uses a lot of resources
Is that still true?

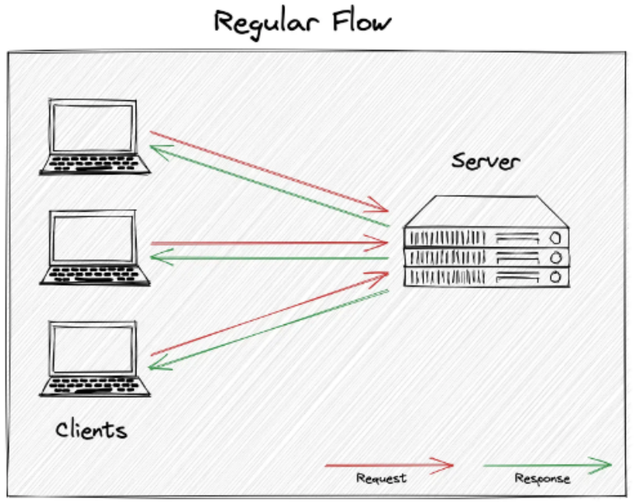
Fear the real enemy ...

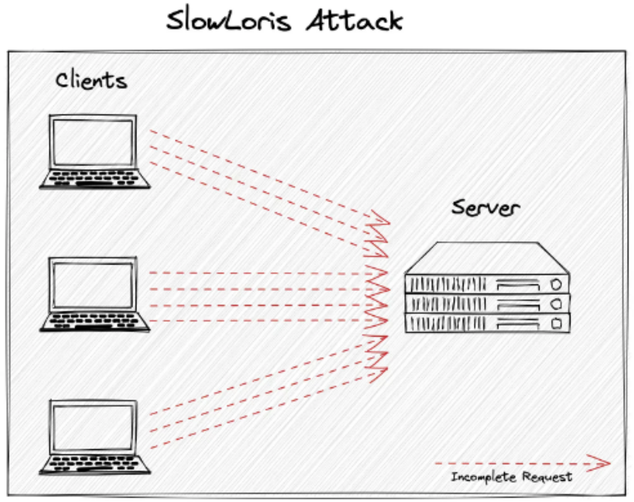
The Slowloris attack

What is it?
A DDoS attack which
uses minimal bandwidth.

When it was created?
Robert “RSnake” Hansen
carved it on June 2009.

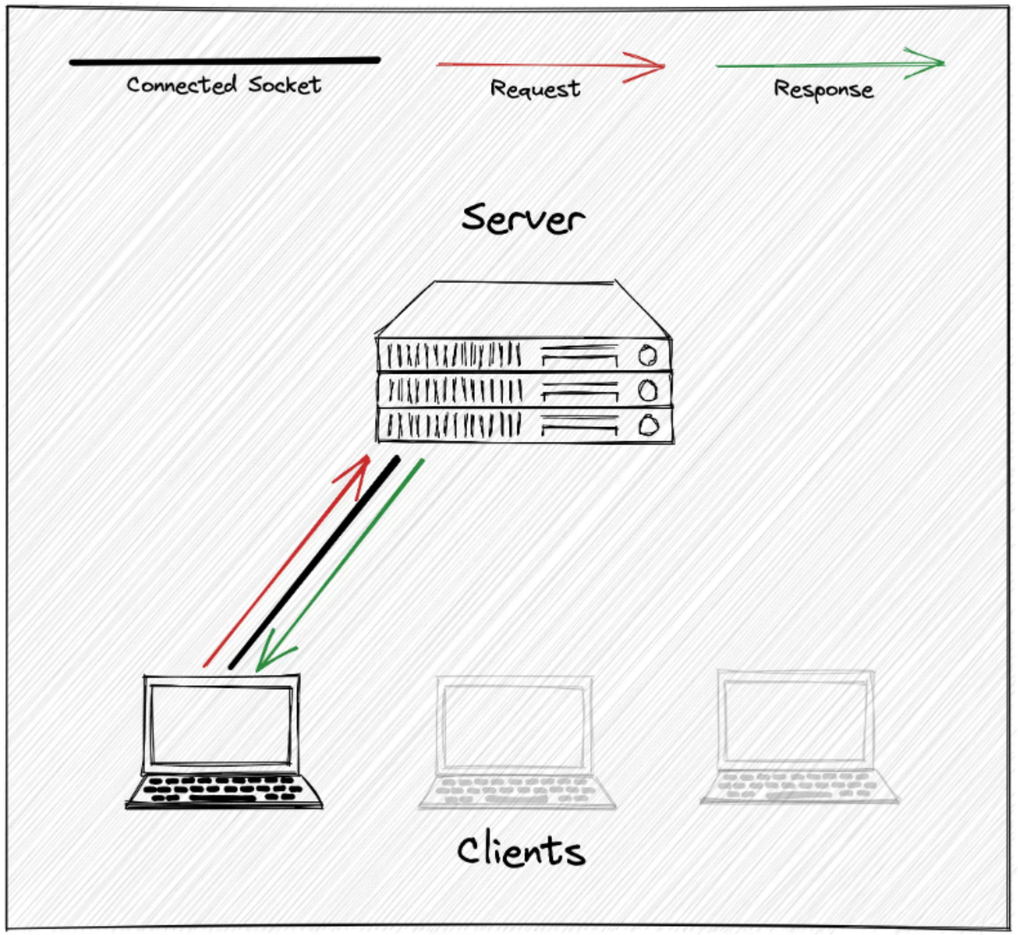
It’s not just HTTP!
This attack can be abstracted
to similar protocols.





Normal HTTP server activity

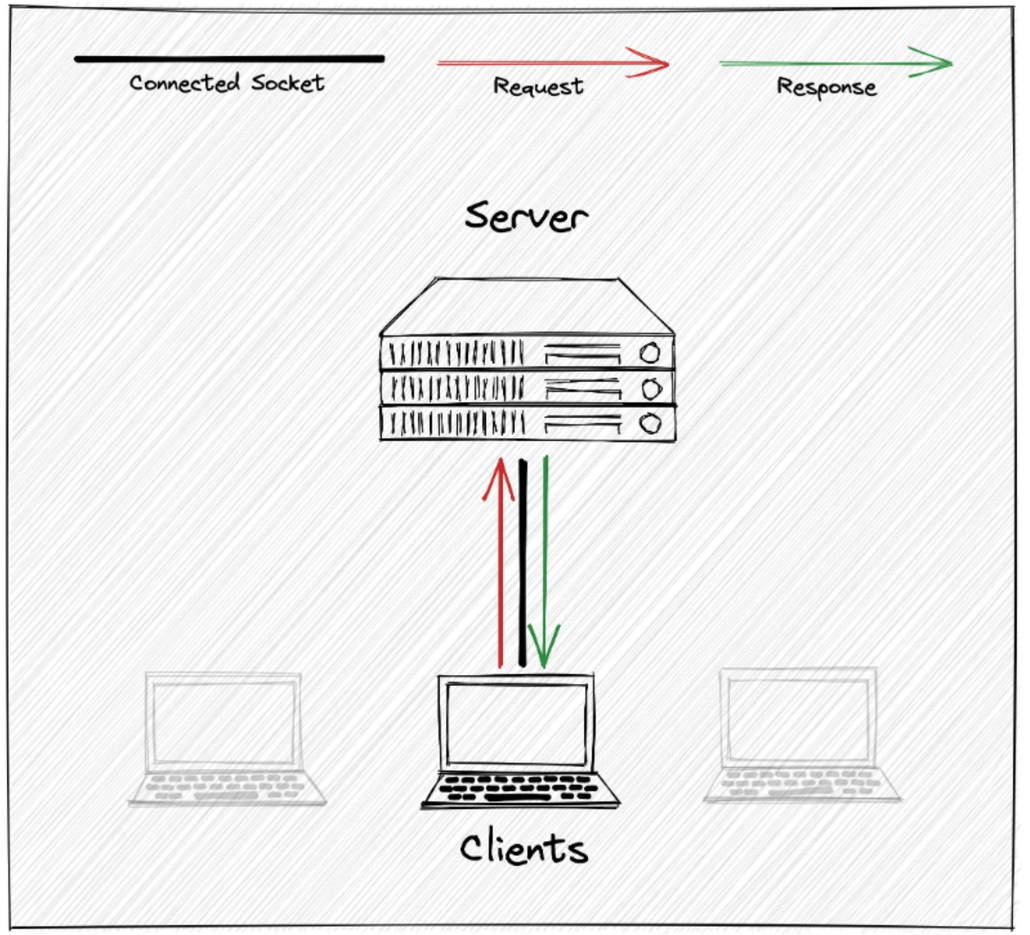
Each socket on the server consumes some amount of RAM and other system resources.



1

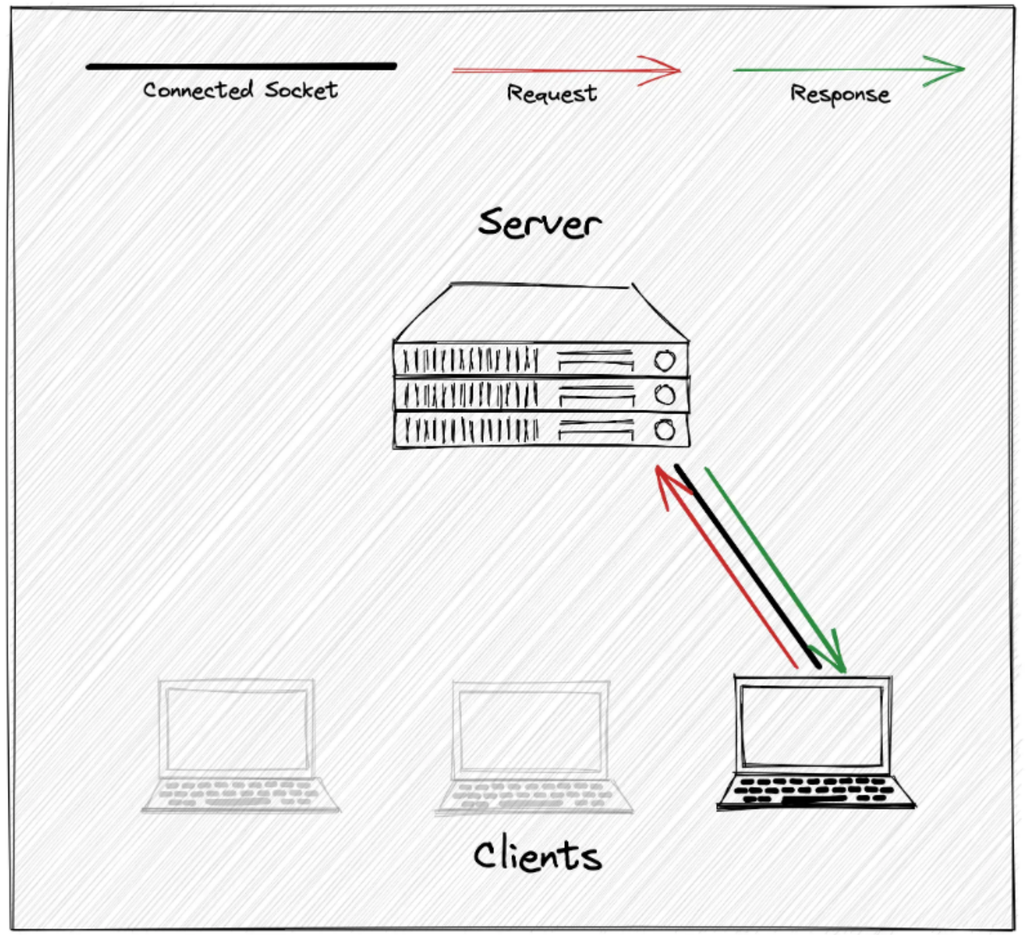
Normal HTTP server activity

Clients usually disconnect after receiving the response.



Normal HTTP server activity

The amount of resources consumed by the server are relatively stable.



Retaining sockets is expensive

The operating system manages low-level operations

Each sockets consumes several kilobytes of RAM.

Each socket is backed by a file descriptor

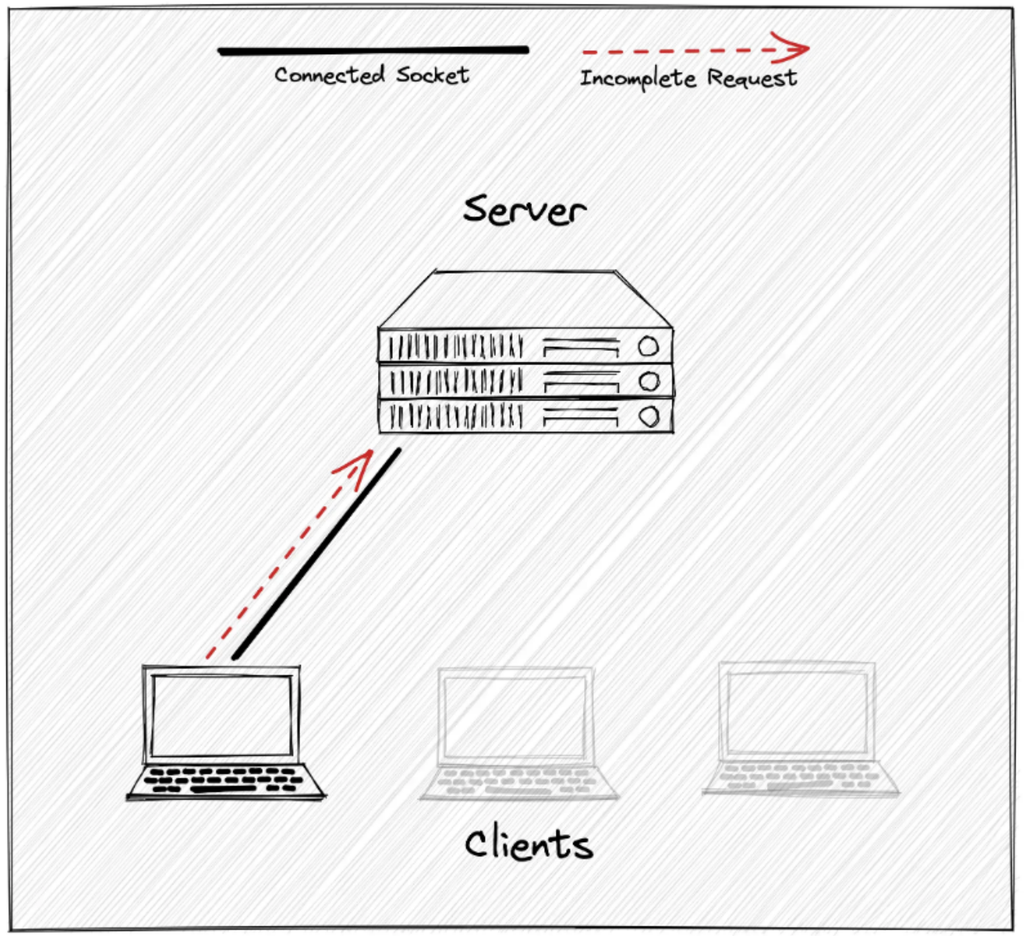
Each process has a limited number of descriptor available, managed via `ulimit`.

The application manages high-level operations

The application representation of the socket adds extra memory overhead.

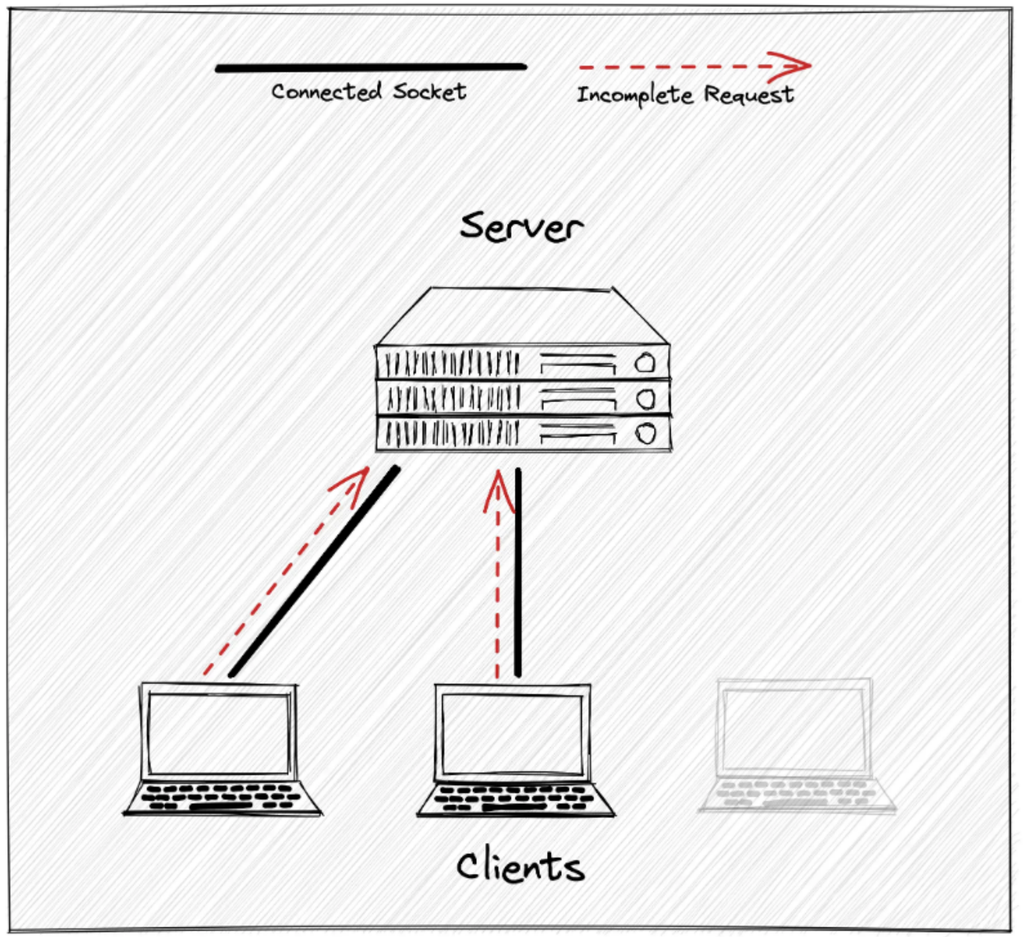
The Slowloris attack

Each client never finishes a request and stays connected for the longest time possible.



The Slowloris attack

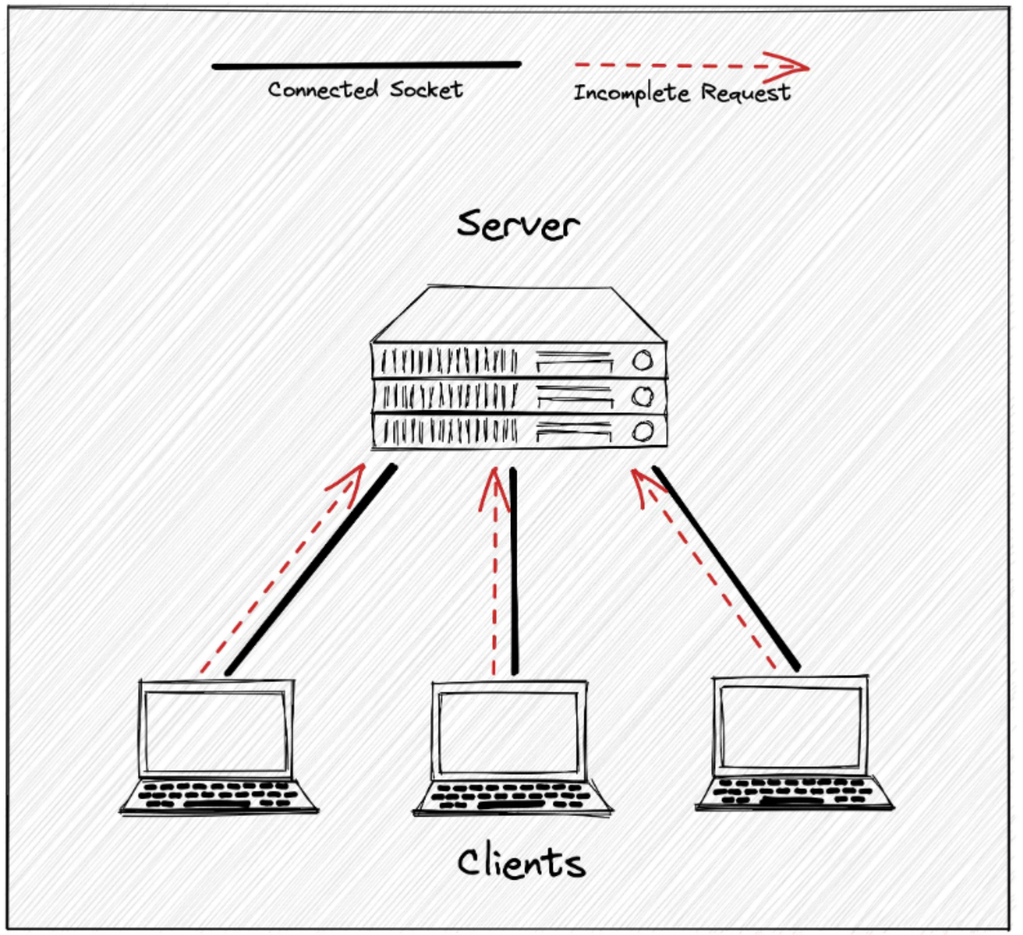
As more clients connect, server resources usage constantly increases over time.



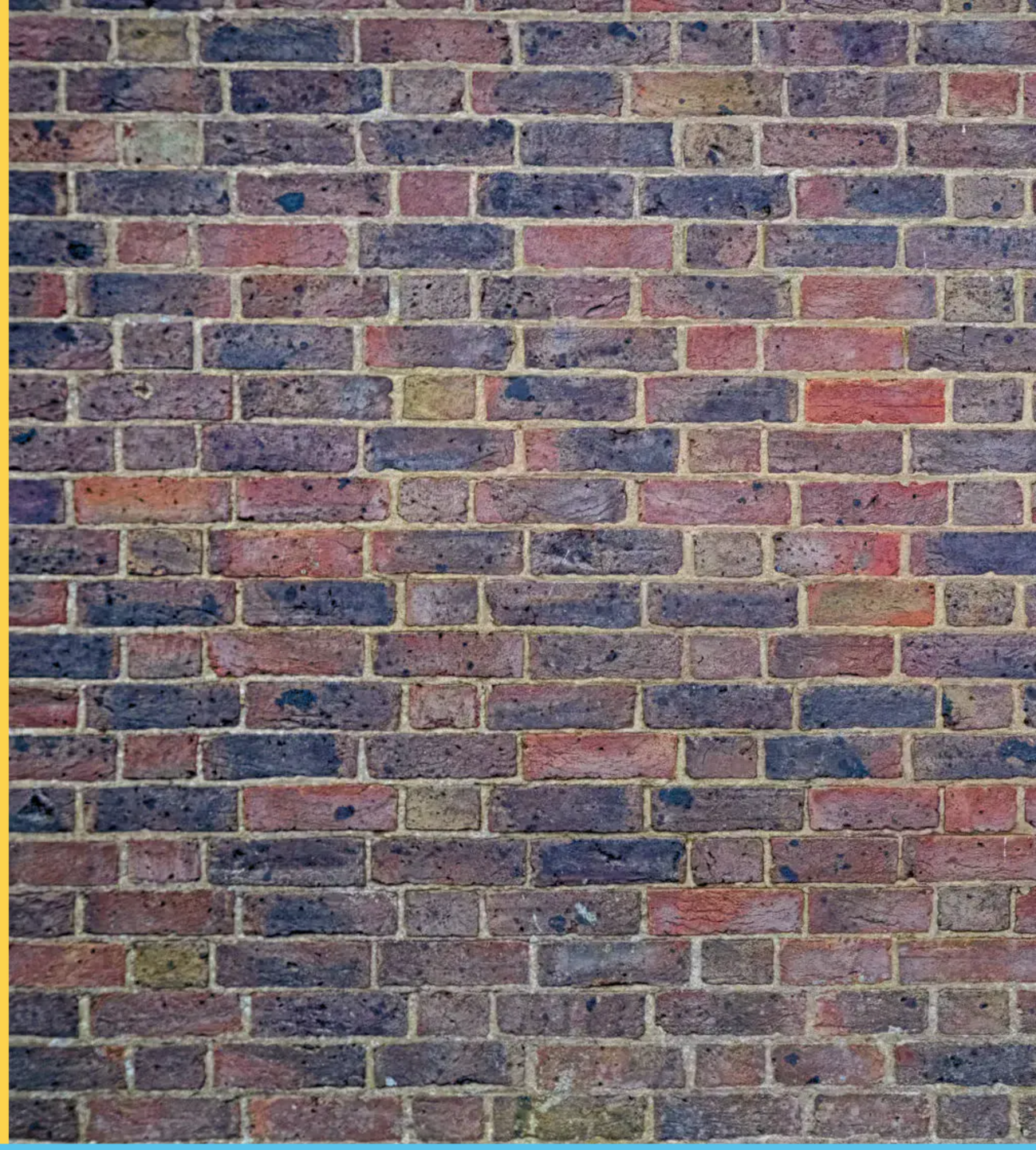
The Slowloris attack

At some point server has no more free resources and cannot accept any new client.

The service is interrupted!



How do we stop it?

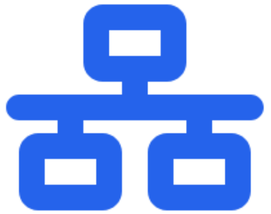


Use a reverse proxy

Never put Node.js as the direct point of contact to the clients.
Servers like Nginx have better protection from DDoS attacks.

Mitigation strategies

Distinguish between requests which are legit and requests that belong to the attack is very hard.
None of the strategies below is 100% accurate.



Limit connections per IP

As the attack is distributed, the attacker can easily switch to another IP.

Enforce speed or time constraints

It's hard to establish a connection which will not cut out slow legit clients.

What about Node.js?

http.Server.headersTimeout

Partial fix as the body is not considered

Node.js never parses or consume request bodies.

Body handling is delegated

Applications are responsible for body timeouts.

Node.js 10.14.0, November 28th, 2018

Node.js has been completely unprotected for 10 years.

10

Trust the frameworks

Disable http.Server.timeout by default
The default value has been changed to nothing.

Node.js 13.0.0, October 22th, 2019
From now on, attacker can delay transfer indefinitely.

The philosophy behind this choice was to **support** serverless environments which needed long running connections.

http.Server.requestTimeout

Complete fix

The client has now limited time to finish.

Available on all active Node.js lines

Added to Node.js 14.11.0, released on September 15th, 2020.

Disabled by default

Adding a clock for each new request is an expensive operation.

Are we safe now?

Yes, almost!

The countermeasures were loose

Custom configuration was needed to protect adequately

The attacker could still delay data transfer forever without being rejected.

Performance was prioritized

Whenever a timeout has gone off was checked only after new data was received.

How to protect Node.js 16 and below

Make sure sockets cannot be idle

`http.Server.timeout` must be greater than 0 to detect malicious idle sockets.

Limit the total time for each request

`http.Server.requestTimeout` must be greater than 0.

Have a lower timeout for the headers

`http.Server.headersTimeout` should also be set to detect a malicious client earlier.

How to protect Node.js 16 and below

```
import { createServer } from 'node:http'

const server = createServer()

// The socket can be idle for 2 minutes before being terminated
server.timeout = 120000

// The entire request must be completely received by the server within 5 minutes
server.requestTimeout = 300000

// The request's headers must be completely received by the server within a minute
server.headersTimeout = 60000
```

Node.js 18.0.0 is finally safe by default

The latest major version of Node.js has finally solved all the issues described.
Changing timeout handling has improved performance by 2%.

Request are checked periodically
The HTTP server regularly checks for requests which might have timed out.

See the new option:
`connectionsCheckingInterval`

Have safer defaults
The timeouts for headers and request are finally enabled by default.

The default Node.js configuration now protects against SlowLoris.

We made it!

Take home lessons

What can we learn from this long journey?

Security, always

Always think about security implication when implementing new features or fixing bugs.

Sacrifice Performance

Putting performance aside can drive to correct, innovative and eventually performant solutions.

Validate regularly

During regular activities, always check the existing code for hidden flaws or vulnerabilites.

One last thing™

“Never assume the obvious is true.”

William Safire

Thank you!

Paolo Insogna

Node.js TSC, Principal Engineer

@p_insogna

paolo.insogna@platformatic.dev

Platformatic