Streams

A stream is an abstract interface for working with streaming data in Node.js.

Streams can be readable, writable, or both. All streams are instances of EventEmitter.

There are four fundamental stream types within Node.js:

- Writable: streams to which data can be written (for example, fs.createWriteStream()).
- Readable: streams from which data can be read (for example, fs.createReadStream()).
- Duplex: streams that are both Readable and Writable (for example, net.Socket).
- Transform: Duplex streams that can modify or transform the data as it is written and read (for example, zlib.createDeflate()).

We can chain streams together. For example, we can create a readable file stream that read a file and a pipe that stream to another writable file stream, which writes the data to a new file.

```
const fs = require('fs');

const fileReadStream = fs.createReadStream(`${__dirname}/server.js`);
const fileWriteStream = fs.createWriteStream(
    `${__dirname}/server_copy.js`,
    { flags: 'wx' }
);

fileReadStream.pipe(fileWriteStream);
```

The Basic Streaming HTTP Server

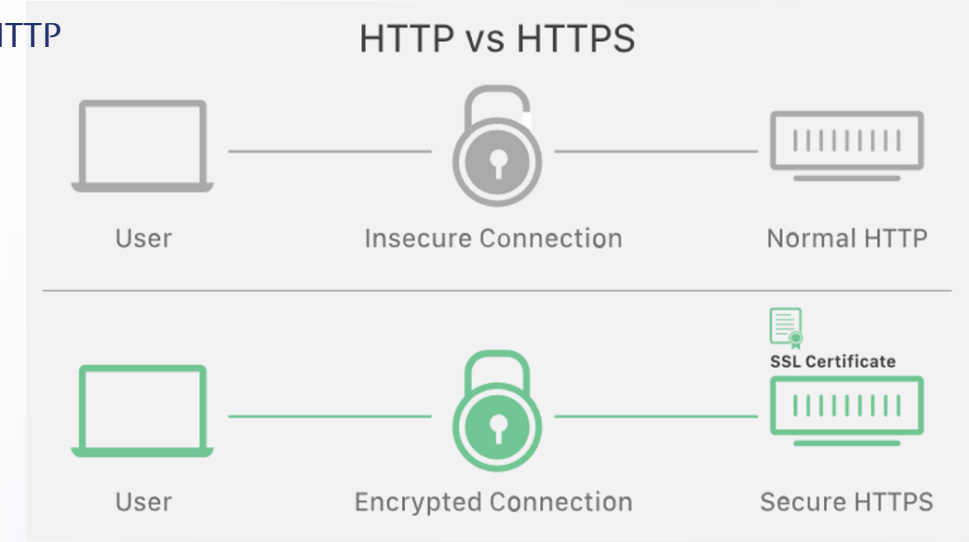
Express's request and response objects are readable and writable streams accordingly.

Let's create an HTTP streaming server that read from a file and send file content as a response.

```
const fs = require('fs');
const express = require('express');

const app = express();

app.get('/', async (req, res) => {
    const fileStream = fs.createReadStream(`${__dirname}/server.js`);
    fileStream.pipe(res);
});

app.listen(3000)
```

HTTPS server

The only difference between the two protocols is that HTTPS uses TLS (SSL) to encrypt normal HTTP requests and responses, and to digitally sign those requests and responses.

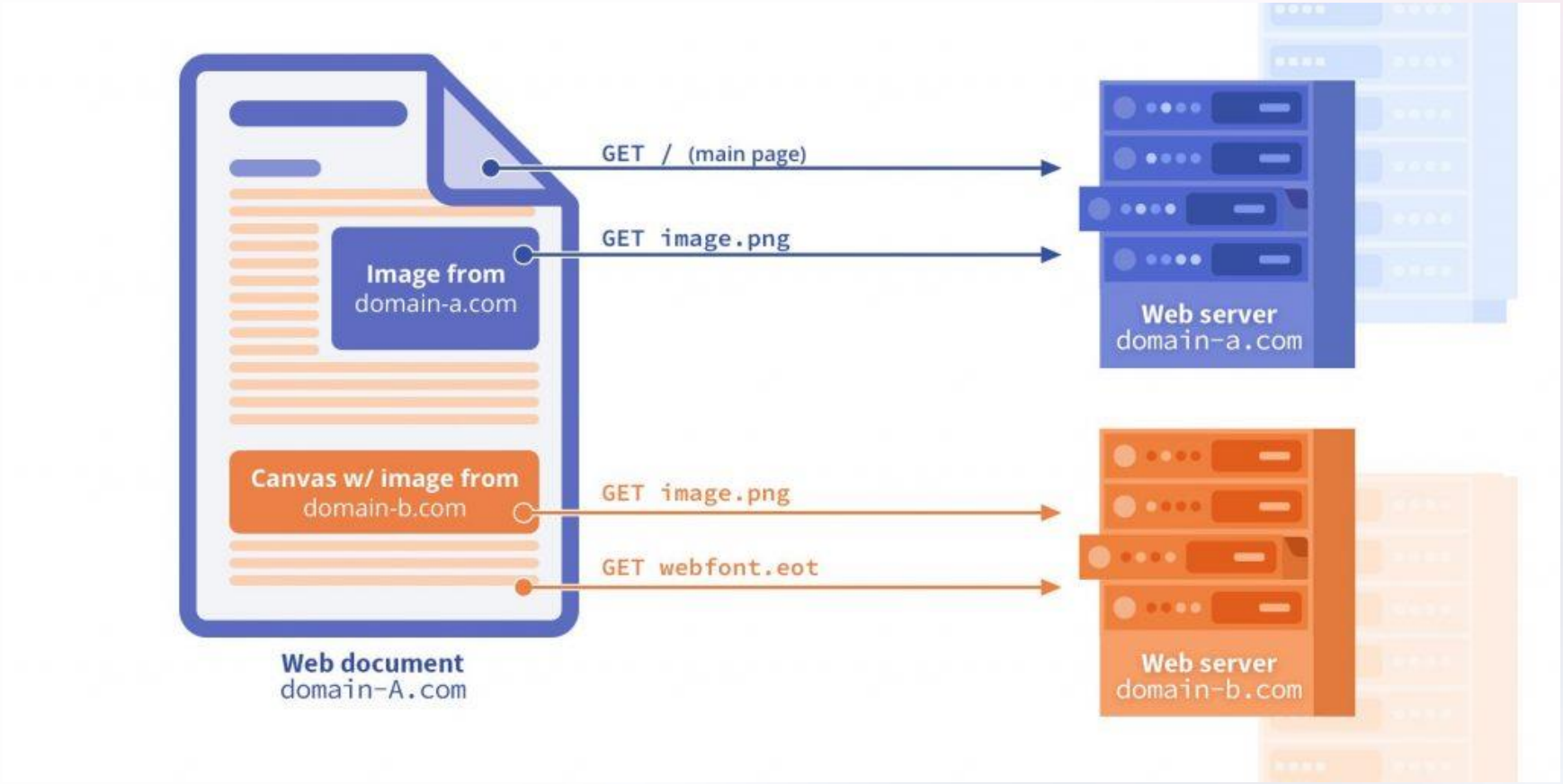



```
const https = require("https");
const fs = require("fs");
const express = require("express");

const app = express();

app.get('/', (req, res) => {
    res.send('Hello')
});

https
    .createServer(
        {
            key: fs.readFileSync("key.pem"),
            cert: fs.readFileSync("cert.pem"),
        },
        app
    )
    .listen(3000);
```

sourcemind

CORS

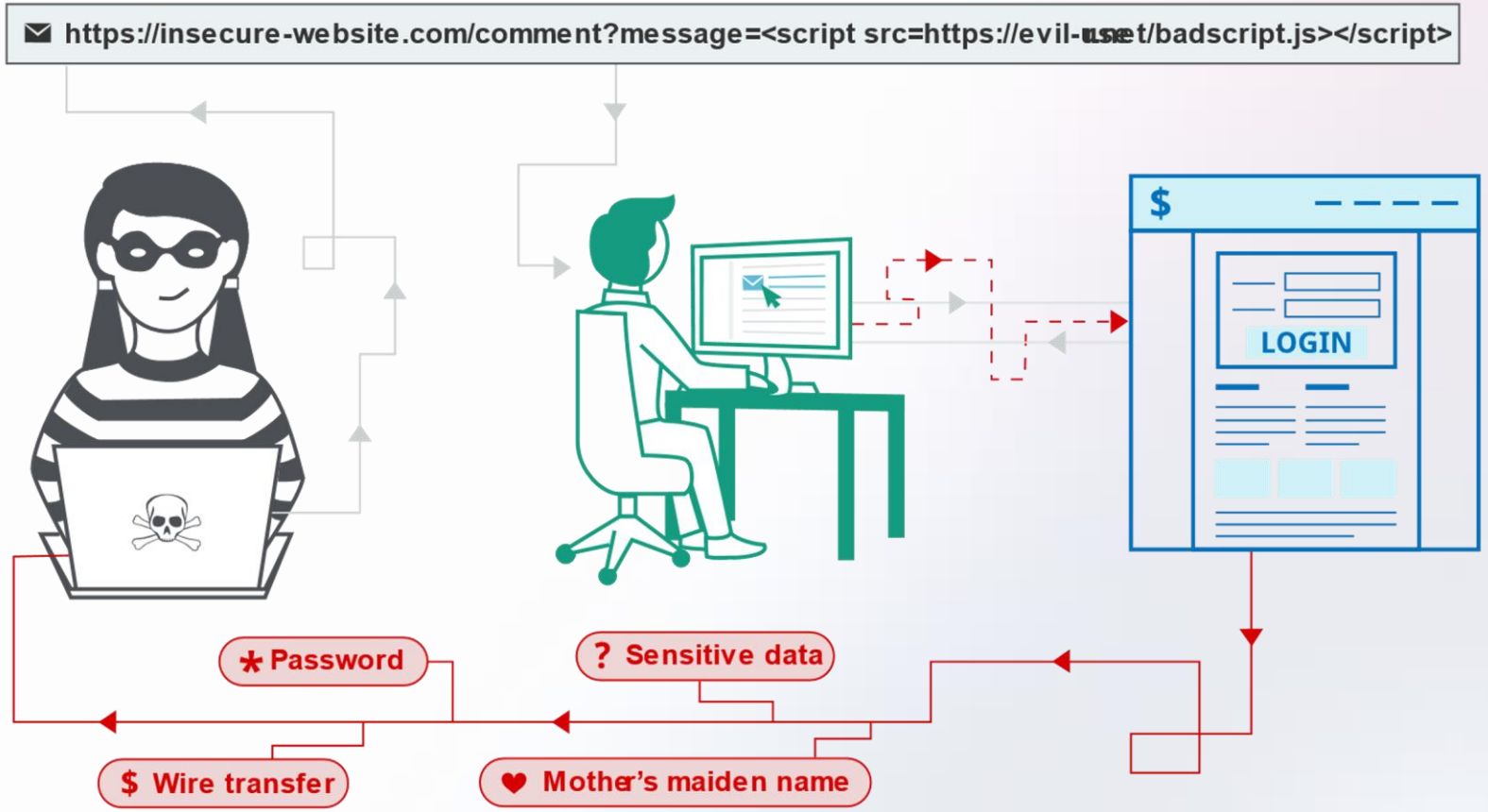
Cors stands for Cross-origin resource sharing. **Cross-origin resource sharing** (**CORS**) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.



sourcemind

XSS - cross-Site Scripting

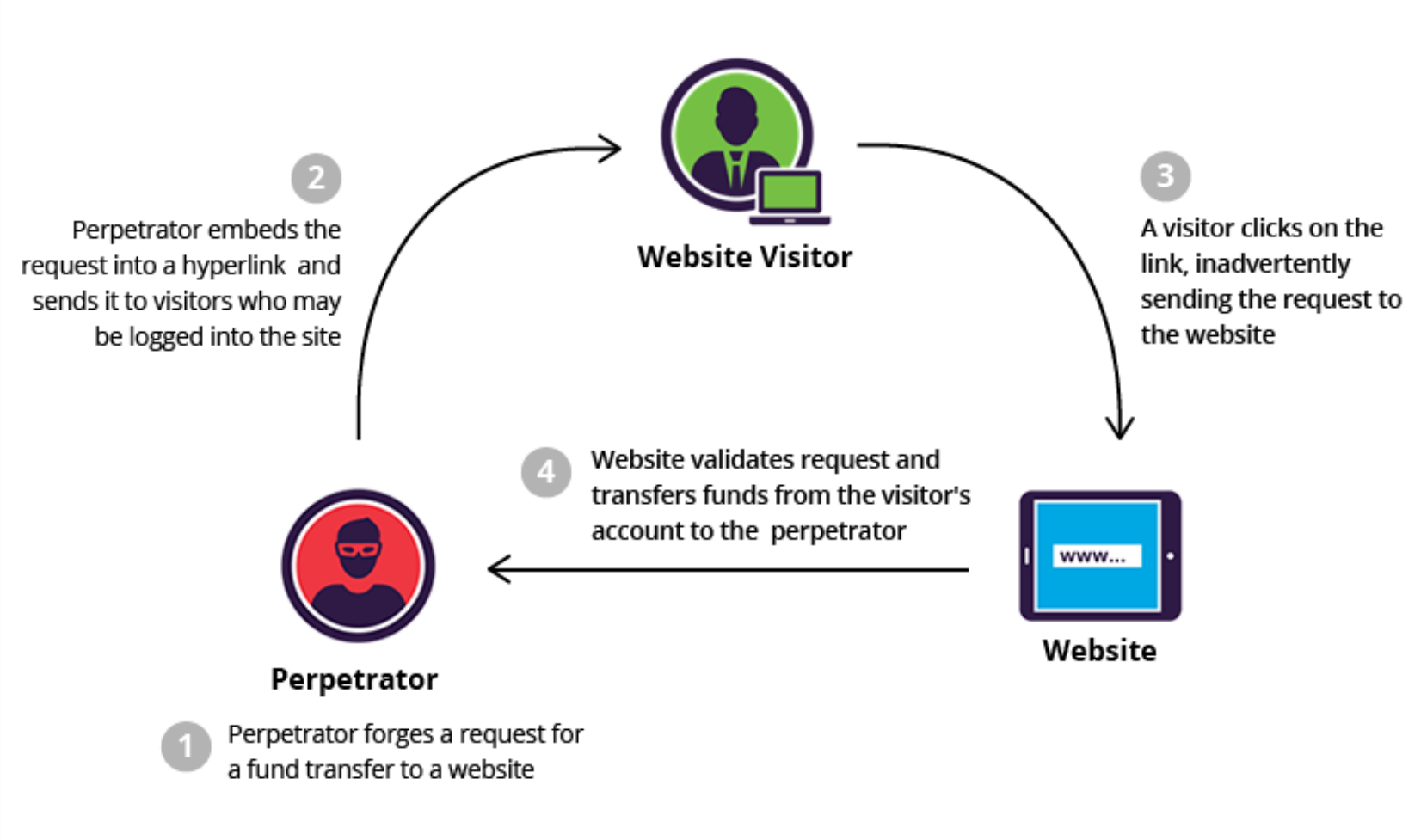
Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.



sourcemind

CSRF

Cross site request forgery (CSRF) is a vulnerability where an attacker performs actions while impersonating another user. For example, **transferring funds to an attacker's account, changing a victim's email address, or they could even just redirect a pizza to an attacker's address!**



sourcemind