



chat on [gitter](#)

Osgood is a secure, fast, and simple platform for running JavaScript HTTP servers. It is written using Rust and V8.

Services written today share a common flaw: Being over-privileged. Osgood is an attempt to build a platform from the ground up, one which applies the *Principle of Least Privilege* at its very core. Osgood requires that policies be written ahead of time describing the I/O requirements of an application. If such an operation hasn't been whitelisted, it will fail. Developers familiar with JavaScript development in the web browser should feel right at home with the APIs provided in Osgood.

Documentation

- [Osgood API Docs](#)
- [Introducing Osgood \(blog\)](#)
- [Hosting a Static Site and Contact Form with Osgood \(blog\)](#)
- [Osgood and CouchDB \(blog\)](#)
- [Introducing Osgood \(presentation\)](#)

Hello, World!

```
1 // app.js
2 app.port = 3000;
3
4 app.get('/hello', 'hello-worker.js');
```

```
1 // hello-worker.js
2 export default () => 'Hello, World!';
```

```
1 $ osgood app.js
2 $ curl http://localhost:3000/hello
```

What is Osgood?

Osgood is a JavaScript runtime purpose-built to run HTTP servers. Its goals are to provide a secure way to build HTTP servers that are fast and simple. Osgood handles server routing and configuration for you, allowing you to focus on application code.

Today we build web applications with general purpose language runtimes. Osgood is an experiment that asks the question: “What if we built a runtime specifically for web apps? What kind of benefits can we get from being at a higher level of abstraction?”

Since the Osgood runtime has intimate knowledge of the routing table we get the ability to isolate controllers for free (we refer to these as Workers). The I/O performed by the application, as well as policy enforcement, happens in Rust-land. Each worker has its own set of permissions.

Here’s an example policy:

```
1 policy.outboundHttp.allowGet('https://intrinsic.com');
```

Consider the situation where Controller A has permission to send a message to `intrinsic.com`, and Controller B has access to user credentials. Within a properly configured Osgood application this means it’s not possible to transmit user credentials to `intrinsic.com`.

Installing Osgood

Download a Prebuilt Release

All prebuilt releases can be downloaded from the Releases page.

Building Osgood

We have more information on compiling Osgood on our Building Osgood wiki page.

Osgood Overview

Application File

An Osgood application file is essentially the entrypoint for the application. Each application will have a single application file. It is the only necessary argument for the `osgood` command.

This file has three purposes:

- Configure global settings such as port and interface
- Route incoming requests to the desired Osgood worker
- Configure the security policies for each Osgood worker

More information about Osgood application files are available on the Osgood Application File wiki page.

Worker File

An Osgood worker file works by exporting a default function. Typically you'll export an [async](#) function but it also works fine by returning a promise or a string value.

Workers are called with information about the incoming request and the returned value is then used to dictate the response to the client.

More information about Osgood worker files are available on the [Osgood Worker Files wiki page](#).

Contributing

Contributions are welcome! Please see [CONTRIBUTING.md](#).

License

Osgood uses the MIT License. Please see [LICENSE.txt](#).