

# Deno's Networking and File Permissions Model

Deno uses a novel permissioning model for allowing access outside the runtime sandbox. In this article we'll cover each option for loosening security restrictions supported by the deno CLI.



Alvin Charity

Published August 2, 2022

## Table of contents

### What is Deno?

### Installation

### Deno Permissions

### Conclusion

## What is Deno?

Deno is a secure Typescript runtime that aims to replace NodeJS for JavaScript users. Deno aims to be a secure computing platform, and as such has a permissions model that forces all scripts to be executed within a sandbox. This is done to protect both you, the developer, and any user of your script from running into any of the common security issues found with NodeJS. Deno essentially makes your environment immutable to any changes, unless you explicitly change its permissions.

However, Deno doesn't completely lock you out of your own device. You are able to explicitly set permissions to access environment variables, network access, and file read / write permissions. Deno additionally provides a foreign function interface (FFI) for the C language Application Binary Interface (ABI).

Despite this extra security, Deno does not require explicit permission to import local or remote modules, similarly to NodeJS and NPM. This means you could potentially import malicious code from a third party source. Deno's FFI feature also allows you to run external scripts outside of the default sandbox, allowing for privilege escalation in some cases. Be sure to audit any third party sources before attempting to use them in your codebase.

The Deno CLI provides options to granularly set permissions during runtime. In this article I will discuss several of those options and provide examples for each.

## Installation


Before getting started with any of the examples, be sure to download and install the latest version of Deno from [deno.land](#). Once you're all set up and have added the `.deno` directory to your \$PATH, you can run `deno --help` to see the full list of help options.

The examples below will be using the `deno run` command and you can see the help page for `run` by entering `deno help run` into your terminal.










## Deno Permissions


The options we will explore today include the following permissions

- `--allow-env`
- `--allow-hrtime`
- `--allow-net`
- `--allow-ffi`
- `--allow-read`
- `--allow-run`
- `--allow-write`

Smoke Test /  Run #4298755

Passed - View Details
2 hours ago
Create Issue

- 
Navigate to Linear – A better way to build pr...  
<https://linear.app/>
0:00 
- 
Click on `Log in` link
0:01 
- 
Visit Linear  
<https://linear.app/login>
0:02 
- 
Click on `Continue with Email` button
0:02 
- 
Select `email` textbox and enter value  

0:03 

Tired of flaky end-to-end tests?  
Create **fast** and **reliable** tests for anything that runs in a browser.  
[Learn more](#)

## **--allow-env**

The `allow-env` command authorizes the Deno runtime to use the environment from your terminal setup. This is useful for pointing the Deno script to various parameters that exist in your shell and is very useful for passing API keys to Deno without having to hard code the information into a script.

For example, if you are using Deno to interact with the Amazon Web Services API, you can write your script as usual and reference the API key in the script itself. The script below retrieves the value for the `AWS_API_KEY` environment variable and logs it to the console if it is found.

```
let awsKey = Deno.env.get("AWS_API_KEY");
console.log(awsKey);
```

Once your script is written, you can include the AWS environment variable into the runtime environment of your script by using `--allow-env` like so

```
deno run --allow-env=AWS_API_KEY awsScript.ts
```

The above example will pass the `AWS_API_KEY` environment variable to the `awsScript.ts` file at runtime. If you attempt to run the `awsScript.ts` file without specifying an environment variable to allow, Deno will prompt you to ask if you would like to rerun the script with access to the desired variable.

Note that Deno assumes any environment variable passed to the script will exist in the underlying system. If you attempt to load an environment variable that does not exist, the result will simply be `undefined` when you attempt to use it.

If you choose not to load the variable at this time by typing `deny` Deno will throw a `PermissionDenied` error.

```
let envVar = Deno.env.get("AWS_API_KEY")
               ^
    at Object.opSync (deno:core/01_core.js:170:12)
    at Object.getEnv [as get] (deno:runtime/js/30_os.js:77:17)
    at file:///Users/me/reflect/deno_permissions/permissionsTest.ts:1:23
```

## **--allow-hrtime**

The `allow-hrtime` command provides more control over the printed timestamps output by your script. This is useful to prevent timing attacks which are a method of breaking a cryptographic system by analyzing the length of time it takes the script to process any cryptographic commands. Using higher resolution timing is also useful during performance testing to calculate the time it takes for Deno to execute specific commands.

Deno supports `user timing level 3` which is not available on every runtime. The timing interface and functions exist in Deno's `Performance module`.

## **--allow-net**

Deno's permissions model does not allow network access by default. This means that you are protected from any script that attempts to access a url on your local network or the internet in general.

If you are testing a script, or if you are creating an application that relies on an external database, you can whitelist any local or external resources by passing their urls to the `--allow-net` command.

```
deno run --allow-net=0.0.0.0,192.168.0.1
```

Running the above command will allow access to `0.0.0.0` and `192.168.0.1` on any port. If you want to restrict networking to specific ports, the `allow-net` command also lets you specify ports for even finer grained control of network access.`

```
deno run --allow-net=0.0.0.0:4545
```

You can allow all network access by simply including `--allow-net` with no items passed. Be careful, this will allow any script to run with access to any host and potentially download and run malicious code.

## **--allow-ffi**

The `--allow-ffi` command lets you specify dynamic libraries to load with your script at runtime. The `ffi` in `allow-ffi` stands for **foreign function interface** and Deno uses this to interact with any library that is compatible with the C language application binary interface (ABI). This includes languages like Rust, C/C++, C#, Zig, Nim, and Kotlin (among others).

The Deno foreign function interface allows you to specify a large number of types from your selected foreign language. Find the full list types [on the Deno Foreign Function Interface API documentation](#)

You can find additional information about composing functions to be used with `--allow-ffi` in the [Deno FFI API page](#)

## **--allow-read**

The `--allow-read` permission tells Deno to allow read access to any files passed to the command, individually or as a list. This is useful for restricting access to only files you specify on your (or your users) system.

This is useful for allowing your script to only interact with the specific files needed at runtime, such as configuration files, or any file needed to process with the script.

Say you have a simple `config.ini` file containing user name and password:

```
user = User Name
pass = userpass
```

And the `ini` file above will be read with this simple script:

```
const contents = await Deno.readFile("./config.ini");
console.log(contents);
```

To allow your script to read this `config.ini` file, execute your script using the `--allow-read` option

```
deno run --allow-read config.ini
```

Specify multiple options by supplying a comma separated list of files you would like to interact with.

```
deno run --allow-read=config.ini,resources.txt
```

The `--allow-read` command works for files and directories - simply pass a directory to the `--allow-read` argument to include that directory as a readable resource. Using `--allow-read` to read directories will output an `asyncIterator` used to iterate over the contents in the directory.

```
{
  [Symbol(Symbol.asyncIterator)]: [AsyncGeneratorFunction: [Symbol.asyncIterator]]
}
```

Note that the `--allow-read` command does not allow Deno to read subdirectories that may be located in the parent directory you pass to the permission. For example, if you would like to allow your script to read your `/etc` directory, you can only files within the `/etc` directory, not subfolders. You must use `--allow-read /etc/<subdirectory>` to allow access to `<subdirectory>`.

As with other Deno permissions, if you do not include `--allow-run` when executing the script, Deno will prompt you to run your command again using the `--allow-run` option to include read access to any files referenced in your script. Choosing `yes` to allow access to these files will re-run the Deno command with the files `allow-read` command populated with filenames referenced in your script.

However, if you choose `deny` to prevent Deno from allowing read access to these files, you are presented with a warning.

```
const contents = await Deno.readFile("./config.ini");
                    ^
at async Object.readFile (deno:runtime/js/40_read_file.js:55:20)
```

```
at async file:///Users/me/reflect/deno_permissions/readFileDirs.ts:1:18
```

## –allow-run

If you would like to execute any files with your script, use the `--allow-run` option to allow Deno to execute any external files as part of the runtime. As with `allow-read`, using `--allow-run` will only let your script execute the files specified with the `deno run` command, eg `deno run --allow-run runFile.ts`.

For example, if `runFile.ts` contains the following command

```
console.log("this file was executed using `deno --allow-run runFile.ts`");
```

Executing `deno run --allow-run runFile.ts` at the command line will output the following text

```
this file was executed using `deno --allow-run runFile.ts`
```

`allow-run` also tells Deno that your script can run command line tools as well. If you need to get the **host name** of the machine running your Deno code, you can tell Deno to allow the `hostname` command using the following

```
deno run --allow-run hostname
```

A quick warning – using `--allow-run` will execute any file or command as a subprocess *outside* of the Deno sandbox, meaning that any permissions set at runtime will not apply to the file or command being executed. This is a type of privilege escalation that you must be aware of when using this permission.

## –allow-write

The `allow-write` option tells Deno to allow write operations on any files specified at the command line. As with `--allow-read`, `--allow-write` accepts single file or directory options, or multiple files and directories passed to the command as a comma separated list.

## Conclusion

In this article you learned a bit about why Deno is “secure by default” and how the command line permissions can help keep your system secure. Additionally, I described a few caveats when it comes to Deno’s permissions – you can import 3rd party code without allowing network access, and the `allow-run` permission will execute any command or code as a separate subprocess, outside of the Deno sandbox.

Deno aims to be a Typescript-first programming environment that will potentially replace NodeJs and NPM. To find out more about Deno permissions, head to [the permissions section of the Deno “getting started” docs](#). You can learn more about Deno at [deno.land](#).

## Get started with Reflect today

Create your first test in 2 minutes, no installation or setup required. Accelerate your testing efforts with fast and maintainable test suites without writing a line of code.

[Try Reflect for free](#)

[Schedule a demo →](#)



Product  
Test Better

Solutions  
Web Application Testing

Company  
About

Documentation  
Overview