MyWebUniversity
python
Red Hat Enterprise Linux Manual Pages
FPDF Library PDF generator
FPDF
Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'npm-audit.1' command

$ man npm-audit.1

NPM-AUDIT(1)   NPM-AUDIT(1)

NAME

npm-audit - Run a security audit

Synopsis

npm audit [fix|signatures]

Description

The audit command submits a description of the dependencies configured in your project to your default registry and asks for a report of known vulnerabilities. If any vulnerabilities are found, then the impact and appropriate remediation will be calculated. If the fix argument is provided, then remediations will be applied to the package tree.

The command will exit with a 0 exit code if no vulnerabilities were found.

Note that some vulnerabilities cannot be fixed automatically and will require manual intervention or review. Also note that since npm audit fix runs a full-fledged npm install under the hood, all configs that apply to the installer will also apply to npm install -- so things like npm audit fix --package-lock-only will work as expected.

By default, the audit command will exit with a non-zero code if any vulnerability is found. It may be useful in CI environments to include the --audit-level parameter to specify the minimum vulnerability level that will cause the command to fail. This option does not filter the report output, it simply changes the command's failure threshold.

Audit Signatures

To ensure the integrity of packages you download from the public npm registry, or any registry that supports signatures, you can verify the registry signatures of downloaded packages using the npm CLI.

Registry signatures can be verified using the following audit command:

```
$ npm audit signatures
```

The npm CLI supports registry signatures and signing keys provided by any registry if the following conventions are followed:

1. Signatures are provided in the package's packument in each published version within the dist object:

```
"dist":{
  "..omitted..": "..omitted..",
  "signatures": [{
    "keyid": "SHA256:{{SHA256_PUBLIC_KEY}}",
    "sig": "a312b9c3cb4a1b693e8ebac5ee1ca9cc01f2661c14391917dcb111517f72370809..."
  }]
}
```

See this example https://registry.npmjs.org/light-cycle/1.4.3 of a signed package from the public npm registry.

The sig is generated using the following template: ${pack? age.name}@${package.version}:${package.dist.integrity} and the keyid has to match one of the public signing keys below.

1. Public signing keys are provided at registry-host.tld/-/npm/v1/keys in the following format:

```
{
  "keys": [{
    "expires": null,
    "keyid": "SHA256:{{SHA256_PUBLIC_KEY}}",
    "keytype": "ecdsa-sha2-nistp256",
    "scheme": "ecdsa-sha2-nistp256",
    "key": "{{B64_PUBLIC_KEY}}"
  }]
}
```

Keys response:

? expires: null or a simplified extended <a
  href="https://en.wikipedia.org/wiki/ISO_8601" target="_blank">ISO
  8601 format</a>: YYYY-MM-DDTHH:mm:ss.sssZ

? keydid: sha256 fingerprint of the public key

? keytype: only ecdsa-sha2-nistp256 is currently supported by the npm
  CLI

? scheme: only ecdsa-sha2-nistp256 is currently supported by the npm
  CLI

? key: base64 encoded public key

See this <a href="https://registry.npmjs.org/-/npm/v1/keys" tar?
get="_blank">example key's response from the public npm registry</a>.

Audit Endpoints

There are two audit endpoints that npm may use to fetch vulnerability
information: the Bulk Advisory endpoint and the Quick Audit endpoint.

Bulk Advisory Endpoint

As of version 7, npm uses the much faster Bulk Advisory endpoint to op?
timize the speed of calculating audit results.

npm will generate a JSON payload with the name and list of versions of
each package in the tree, and POST it to the default configured reg?
istry at the path /-/npm/v1/security/advisories/bulk.

Any packages in the tree that do not have a version field in their
package.json file will be ignored. If any --omit options are specified
(either via the --omit config, or one of the shorthands such as --pro?
duction, --only=dev, and so on), then packages will be omitted from the
submitted payload as appropriate.

If the registry responds with an error, or with an invalid response,
then npm will attempt to load advisory data from the Quick Audit end?
point.

The expected result will contain a set of advisory objects for each de?
pendency that matches the advisory range. Each advisory object con?
tains a name, url, id, severity, vulnerable_versions, and title.

npm then uses these advisory objects to calculate vulnerabilities and meta-vulnerabilities of the dependencies within the tree.

Quick Audit Endpoint

If the Bulk Advisory endpoint returns an error, or invalid data, npm will attempt to load advisory data from the Quick Audit endpoint, which is considerably slower in most cases.

The full package tree as found in package-lock.json is submitted, along with the following pieces of additional metadata:

? npm_version

? node_version

? platform

? arch

? node_env

All packages in the tree are submitted to the Quick Audit endpoint. Omitted dependency types are skipped when generating the report.

Scrubbing

Out of an abundance of caution, npm versions 5 and 6 would "scrub" any packages from the submitted report if their name contained a / charac? ter, so as to avoid leaking the names of potentially private packages or git URLs.

However, in practice, this resulted in audits often failing to properly detect meta-vulnerabilities, because the tree would appear to be in? valid due to missing dependencies, and prevented the detection of vul? nerabilities in package trees that used git dependencies or private modules.

This scrubbing has been removed from npm as of version 7.

Calculating Meta-Vulnerabilities and Remediations

npm uses the @npmcli/metavuln-calculator http://npm.im/@npm? cli/metavuln-calculator module to turn a set of security advisories into a set of "vulnerability" objects. A "meta-vulnerability" is a de? pendency that is vulnerable by virtue of dependence on vulnerable ver? sions of a vulnerable package.

For example, if the package foo is vulnerable in the range >=1.0.2 <2.0.0, and the package bar depends on foo@^1.1.0, then that version of bar can only be installed by installing a vulnerable version of foo.

In this case, bar is a "metavulnerability".

Once metavulnerabilities for a given package are calculated, they are cached in the ~/.npm folder and only re-evaluated if the advisory range changes, or a new version of the package is published (in which case, the new version is checked for metavulnerable status as well).

If the chain of metavulnerabilities extends all the way to the root project, and it cannot be updated without changing its dependency ranges, then npm audit fix will require the --force option to apply the remediation. If remediations do not require changes to the dependency ranges, then all vulnerable packages will be updated to a version that does not have an advisory or metavulnerability posted against it.

Exit Code

The npm audit command will exit with a 0 exit code if no vulnerabili? ties were found. The npm audit fix command will exit with 0 exit code if no vulnerabilities are found or if the remediation is able to suc? cessfully fix all vulnerabilities.

If vulnerabilities were found the exit code will depend on the au? dit-level configuration setting.

Examples

Scan your project for vulnerabilities and automatically install any compatible updates to vulnerable dependencies:

```
$ npm audit fix
```

Run audit fix without modifying node_modules, but still updating the pkglock:

```
$ npm audit fix --package-lock-only
```

Skip updating devDependencies:

```
$ npm audit fix --only=prod
```

Have audit fix install SemVer-major updates to toplevel dependencies, not just SemVer-compatible ones:

```
$ npm audit fix --force
```

Do a dry run to get an idea of what audit fix will do, and also output install information in JSON format:

```
  $ npm audit fix --dry-run --json
```

Scan your project for vulnerabilities and just show the details, with?

out fixing anything:

```
  $ npm audit
```

Get the detailed audit report in JSON format:

```
  $ npm audit --json
```

Fail an audit only if the results include a vulnerability with a level

of moderate or higher:

```
  $ npm audit --audit-level=moderate
```

Configuration

audit-level

? Default: null

? Type: null, "info", "low", "moderate", "high", "critical", or "none"

The minimum level of vulnerability for npm audit to exit with a

non-zero exit code.

dry-run

? Default: false

? Type: Boolean

Indicates that you don't want npm to make any changes and that it

should only report what it would have done. This can be passed into any

of the commands that modify your local installation, eg, install, up?

date, dedupe, uninstall, as well as pack and publish.

Note: This is NOT honored by other network related commands, eg

dist-tags, owner, etc.

force

? Default: false

? Type: Boolean

Removes various protections against unfortunate side effects, common

mistakes, unnecessary performance degradation, and malicious input.

? Allow clobbering non-npm files in global installs.

? Allow the npm version command to work on an unclean git repository.

? Allow deleting the cache folder with npm cache clean.

? Allow installing packages that have an engines declaration requiring a different version of npm.

? Allow installing packages that have an engines declaration requiring a different version of node, even if --engine-strict is enabled.

? Allow npm audit fix to install modules outside your stated dependency range (including SemVer-major changes).

? Allow unpublishing all versions of a published package.

? Allow conflicting peerDependencies to be installed in the root project.

? Implicitly set --yes during npm init.

? Allow clobbering existing values in npm pkg

? Allow unpublishing of entire packages (not just a single version).

If you don't have a clear idea of what you want to do, it is strongly recommended that you do not use this option!

json

? Default: false

? Type: Boolean

Whether or not to output JSON data, rather than the normal output.

? In npm pkg set it enables parsing set values with JSON.parse() before saving them to your package.json.

Not supported by all npm commands.

package-lock-only

? Default: false

? Type: Boolean

If set to true, the current operation will only use the pack?
age-lock.json, ignoring node_modules.

For update this means only the package-lock.json will be updated, in?
stead of checking node_modules and downloading dependencies.

For list this means the output will be based on the tree described by the package-lock.json, rather than the contents of node_modules.

omit

? Default: 'dev' if the NODE_ENV environment variable is set to 'pro?
duction', otherwise empty.

? Type: "dev", "optional", or "peer" (can be set multiple times)

Dependency types to omit from the installation tree on disk.

Note that these dependencies are still resolved and added to the pack?
age-lock.json or npm-shrinkwrap.json file. They are just not physically
installed on disk.

If a package type appears in both the --include and --omit lists, then
it will be included.

If the resulting omit list includes 'dev', then the NODE_ENV environ?
ment variable will be set to 'production' for all lifecycle scripts.

foreground-scripts

? Default: false

? Type: Boolean

Run all build scripts (ie, preinstall, install, and postinstall)
scripts for installed packages in the foreground process, sharing stan?
dard input, output, and error with the main npm process.

Note that this will generally make installs run slower, and be much
noisier, but can be useful for debugging.

ignore-scripts

? Default: false

? Type: Boolean

If true, npm does not run scripts specified in package.json files.

Note that commands explicitly intended to run a particular script, such
as npm start, npm stop, npm restart, npm test, and npm run-script will
still run their intended script if ignore-scripts is set, but they will
not run any pre- or post-scripts.

workspace

? Default:

? Type: String (can be set multiple times)

Enable running a command in the context of the configured workspaces of
the current project while filtering by running only the workspaces de?
fined by this configuration option.

Valid values for the workspace config are either:

? Workspace names

? Path to a workspace directory

? Path to a parent workspace directory (will result in selecting all
  workspaces within that folder)

When set for the npm init command, this may be set to the folder of a workspace which does not yet exist, to create the folder and set it up as a brand new workspace within the project.

This value is not exported to the environment for child processes.

workspaces

? Default: null

? Type: null or Boolean

Set to true to run the command in the context of all configured workspaces.

Explicitly setting this to false will cause commands like install to ignore workspaces altogether. When not set explicitly:

? Commands that operate on the node_modules tree (install, update, etc.) will link workspaces into the node_modules folder. - Commands that do other things (test, exec, publish, etc.) will operate on the root project, unless one or more workspaces are specified in the workspace config.

This value is not exported to the environment for child processes.

include-workspace-root

? Default: false

? Type: Boolean

Include the workspace root when workspaces are enabled for a command.

When false, specifying individual workspaces via the workspace config, or all workspaces via the workspaces flag, will cause npm to operate only on the specified workspaces, and not on the root project.

This value is not exported to the environment for child processes.

install-links

? Default: false

? Type: Boolean

When set file: protocol dependencies that exist outside of the project root will be packed and installed as regular dependencies instead of creating a symlink. This option has no effect on workspaces.

See Also

? npm help install

? npm help config