EGIRNA TECHNOLOGIES

Phone: +20-11012200404
E-mail: info@egirna.com
Website: www.egirna.com

FreeBSD Port Documentation

Creating a Port to depend upon in our OPNSense plugin.

FreeBSD Ports

FreeBSD is bundled with a rich collection of system tools as part of the base system. In addition, FreeBSD provides two complementary technologies for installing third-party software: the FreeBSD Ports Collection, for installing from source, and packages, for installing from pre-built binaries. Either method may be used to install software from local media or from the network.
The FreeBSD ports and packages collections offer two simple ways for users and administrators to install over 30,000 applications, utilities or libraries.

A FreeBSD port is a collection of files designed to automate the process of compiling an application from source code. The files that comprise a port contain all the necessary information to automatically download, extract, patch, compile, and install the application.

The Ports Collection is a set of Makefiles, patches, and description files. Each set of these files is used to compile and install an individual application on FreeBSD, and is called a port.

By default, the Ports Collection itself is stored as a subdirectory of /usr/ports.

Getting Started with Ports

The Ports Collection contains directories for software categories. Inside each category are subdirectories for individual applications. Each application subdirectory contains a set of files that tells FreeBSD how to compile and install that program, called a ports skeleton. Each port skeleton includes these files and directories:

- **Makefile:** contains statements that specify how the application should be compiled and where its components should be installed.

- **distinfo:** contains the names and checksums of the files that must be downloaded to build the port.
- **files/:** this directory contains any patches needed for the program to compile and install on FreeBSD. This directory may also contain other files used to build the port.
- **pkg-descr:** provides a more detailed description of the program.
- **pkg-plist:** a list of all the files that will be installed by the port. It also tells the ports system which files to remove upon deinstallation.

Before an application can be compiled using a port, the Ports Collection must first be installed. If it was not installed during the installation of FreeBSD, use one of the following methods to install it:

1. Git must be installed before it can be used to check out the ports tree. If a copy of the ports tree is already present, install Git like this:
   ```
   # cd /usr/ports/devel/git
   # make install clean
   ```
   If the ports tree is not available, or pkg is being used to manage packages, Git can be installed as a package:
   ```
   # pkg install git
   ```
2. Check out a copy of the HEAD branch of the ports tree:
   ```
   # git clone https://git.FreeBSD.org/ports.git /usr/ports
   ```

To compile and install the port, change to the directory of the port to be installed, then type make install at the prompt. Messages will indicate the progress:

```
# cd /usr/ports/security/icapeg
# make install
```

During installation, a working subdirectory is created which contains all the temporary files used during compilation. Removing this directory saves disk space and minimizes the chance of problems later when upgrading to the newer version of the port:

```
# make clean
```

It's worth noting that OPNSense has their own ports tree:
https://github.com/opnsense/ports

Creating our own Port

First, this is the sequence of events which occurs when the user first types `make` in the port's directory.

1. The `fetch` target is run. The `fetch` target is responsible for making sure that the tarball exists locally in `DISTDIR`. If `fetch` cannot find the required files in `DISTDIR` it will look up the URL `MASTER_SITES`, which is set in the Makefile, as well as our FTP mirrors where we put distfiles as backup. It will then attempt to fetch the named distribution file with `FETCH`, assuming that the requesting site has direct access to the Internet. If that succeeds, it will save the file in `DISTDIR` for future use and proceed.
2. The `extract` target is run. It looks for the port's distribution file (typically a compressed tarball) in `DISTDIR` and unpacks it into a temporary subdirectory specified by `WRKDIR` (defaults to work).
3. The `patch` target is run. First, any patches defined in `PATCHFILES` are applied. Second, if any patch files named patch-* are found in `PATCHDIR` (defaults to the files subdirectory), they are applied at this time in alphabetical order.
4. The `configure` target is run. This can do any one of many different things.
    1. If it exists, scripts/configure is run.
    2. If `HAS_CONFIGURE` or `GNU_CONFIGURE` is set, WRKSRC/configure is run.
5. The `build` target is run. This is responsible for descending into the port's private working directory (`WRKSRC`) and building it.
6. The `stage` target is run. This puts the final set of built files into a temporary directory (`STAGEDIR`, see Staging). The hierarchy of this directory mirrors that of the system on which the package will be installed.
7. The `package` target is run. This creates a package using the files from the temporary directory created during the `stage` target and the port's pkg-plist.

8. The `install` target is run. This installs the package created during the `package` target into the host system.

Let's take them step by step. First for fetching and extracting the Sources we will be using GitHub. For that we have to release a version of icap-eg with its own tag using GitHub releases. After reaching a stable point in development we create a tag or a release.

Then the relevant parts of the Makefile are:

```
USE_GITHUB=             yes
GH_ACCOUNT=             KhaledEmaraDev
PORTVERSION=            0.1.0
DISTVERSIONPREFIX=      v
```

This will pull the release named *v0.1.0* from the https://github.com/KhaledEmaraDev/icapeg repo.

This will download a compressed tar file containing the sources and extract it.

Now let's get onto the building process. Fortunately since we are using golang. FreeBSD has included some nice MACROs to help with installing go programs. The steps for this is a s follow:

1. We include this in the Makefile
   ```
   USES=                   go:modules
   ```
2. Then we run

```
Unset
        make makesum
```

3. After that we run

```
Unset
        make gomod-vendor
```

4. This will result in an output named `GH_TUPLE`. We will copy and paste this into the Makefile.
5. Finally we run again:

```
Unset
        make makesum
```

6. This will generate the `distinfo` file which contains checksums for all the go modules we will be using.

Now let's move to the stage step. Here we want to include two files:

1. The configuration file which ideally is at this standard path that the plugin reads from:
    a. /usr/local/etc/icapeg/config.toml
2. The exception-page.

Both of these are done in the `post-install` step.

Now at last we can include the files that we will install here:

```
PLIST_FILES=    bin/icapeg \
                        "@sample ${ETCDIR}/config.toml.sample" \
                        "${DATADIR}/exception-page.html"
```

Finally we can package and test all this up using:

```
make stage
make install
```

Now we are ready to test our installation.

Creating a patch to submit to FreeBSD.

Now that we have a port we need to publish it to the Ports repo. To do this we will have to create a diff patch and send it to FreeBSD. Here are the steps:

1. `git add .`
2. `git commit`
3. `git format-patch origin/main`

This will create a .patch file that can be emailed over at https://bugs.freebsd.org/bugzilla/enter_bug.cgi

Use product "Ports & Packages", component "Individual Port(s)", and follow the guidelines shown there.

Creating a new revision

In order to create a new version we simply create a new GitHub release then bump

```
PORTVERSION=    0.1.0
```

and repeat the installation and patching steps.