

Methods of installation sandboxing for pkgsrc

Tonnerre Lombard

November 4th, 2005

Abstract

This document describes a collection of methods of `;;sandboxing;;` pkgsrc builds, which means that the installation routine will run normally without any special measures, but the program is still packaged and installed by the package system, which is pkgsrc in this case.

It's no secret that a lot of people strongly dislike the current PLIST approach to defining which packages belong into a certain package. There are various reasons why this approach isn't widely accepted, although it almost always seems to work: For one, it puts humans into the process and thus causes an additional load and requires special knowledge of the way pkgsrc works if you want to build a certain package. Then, it requires a bunch of hacks in order to support all the different package options which may cause problems with files that only exist in some configurations.

For all these reasons (and many others), there have been requests to replace the old packlist based mechanism with something real that can't accidentally neglect packages. Here are some approaches that would take this into account. The basic approach is always to install the package into a sandbox directory first, tar it up and then untar it over the target VFS.

Contents

1	Modifying the pkg_install toolchain	3
1.1	Advantages	3
1.2	Disadvantages	3
2	Using a special library	3
2.1	Advantages	3
2.2	Disadvantages	3
3	Using systrace	4
3.1	Advantages	4
3.2	Disadvantages	4
4	Using union mounts	4
4.1	Advantages	4
4.2	Disadvantages	4
5	Using a pkg_comp jail	5
5.1	Advantages	5
5.2	Disadvantages	5
6	Tuning pkg_comp with MD5 sums	5
6.1	Advantages	5
6.2	Disadvantages	5
7	Conclusions	6
A	Author	6
B	Example implementations	6

1 Modifying the `pkg_install` toolchain

This is basically the most natural approach to `pkgsrc`, as it is the one we already use in order to compile packages (where we simply push another `gcc` via the `PATH` variable et al). In this case, the `install(1)`, `install-info(1)` etc. programs will be replaced with simple versions that install to a jail instead, just like eventual `chmod(1)` calls etc. This should do for most applications.

1.1 Advantages

- Simple approach, not hard to install
- Requires no privileges
- Logical conclusion from the other implementations

1.2 Disadvantages

- Doesn't apply to packages that come with their own binary installer
- Broken packages that install files during the build phase will cause problems
- Easy to lose track of files unless the installation process is 100% sober

2 Using a special library

The *Gentoo* approach would be to preload a library that overrides some `libc` calls, modifies the function parameters and then calls the original `libc` functions.

2.1 Advantages

- Doesn't require additional privileges
- Also applies to a couple of binary installers (But not all of them)

2.2 Disadvantages

- Doesn't work for binary installers that circumvent `libc`
- It's hard to tell if a touched file really belongs to the packet or if it's just being looked at
- Problems with `SetUID` environments

3 Using systrace

This is basically the *OpenBSD* method. The package installation routine runs under surveillance by `systrace(1)`, which will then modify the target paths for file accesses so the package basically ends up somewhere else, while the installation routine still believes to have done a pretty good job.

3.1 Advantages

- Existing prior art (OpenBSD ports)
- Catches the problem by its roots, can't lose a file access
- Also works for weird binary installers that circumvent libc

3.2 Disadvantages

- It's hard to tell if a touched file really belongs to the packet or if it's just being looked at
- Requires special privileges
- Not 100% portable

4 Using union mounts

This is probably the most clean solution so far. The root file system is mounted read-only into a subdirectory, and a temporary file system is mounted over it for writing using union mounts. All files appear in their usual place for the installation routine, but all write accesses end up in a temporary file system that you can easily tar up after the installation routine has finished.

4.1 Advantages

- Takes care of all possible cases
- Only sandboxes files that really changed
- Also takes care of build fugitives

4.2 Disadvantages

- Not very portable
- Requires special privileges

5 Using a pkg_comp jail

This is basically the current clean-room approach pkgsrc is taking. A second root file system is installed into a chroot jail, and all packages installed into it get tracked by the print-PLIST mechanism, which basically walks through the file list and prints out the name of every file that doesn't belong to a package yet.

5.1 Advantages

- Already implemented
- Very portable
- Also takes care of build fugitives and binary installers

5.2 Disadvantages

- Takes a lot of time and expertise to set up
- Requires special privileges
- Can't provide automatic zero-clue installation of packages
- Doesn't take changed files from other packages into account

6 Tuning pkg_comp with MD5 sums

The idea behind this method would be to extend the package lists with MD5 sums of every file the package contains. The print-PLIST mechanism would then check the MD5 sum of every file in the target and if the file was modified, it will be taken into the list of files of the package. Just like its original, this method is only useful in dedicated pkg_comp jails.

6.1 Advantages

- Easy to implement
- Very portable
- Also takes care of build fugitives and binary installers
- Also takes care of changed files from other packages

6.2 Disadvantages

- Takes a lot of time and expertise to set up
- Requires special privileges
- Can't provide automatic zero-clue installation of packages

7 Conclusions

There doesn't appear to be a perfect method that doesn't have flaws and is portable. The quality of the approaches is an inverse function of their portability, which is sad. However, it is still possible to implement a number of these methods and use more portable ones as a fallback if the underlying operating system doesn't support it.

My suggestion is to use the `systrace` approach if running on a BSD system and to keep the `PLIST` files around as a fallback solution. That way, most of the likely `pkgsrc` users will have the new features while the few ones that try to use `pkgsrc` on other platforms still can do that.

Also, different approaches can be applied to different packages. Since most packages only use the `pkg_install` framework, the modified toolchain will do perfectly well for them while still making for a portable package. Only special cases need tricks like `systrace` at all in the first place, such as Nullsoft Java installers and friends.

A Author

Tonnerre Lombard is a cross-system kernel hacker and political activist. He has founded the FFII Switzerland and is trying to get a political movement for human rights and free communication into place.

You can contact the author via:

Email tonnerre@thundrix.ch (HTML mail will be **DELETED!**)

Jabber tonnerre@jabber.org

Postal service Don't contact me this way, please!

You can visit the author's homepage on <http://users.thundrix.ch/~tonnerre/>.

B Example implementations

There are no example implementations for `pkgsrc` known yet. However, some of the designs have prior art:

`pkg_install` The changed `pkg_install` is similar to the `pkgsrc` wrappers.

`libsandbox` The sandbox library approach has been previously implemented in Gentoo Linux, but doesn't always work well.

`systrace` The **systrace**(1) approach is derived from OpenBSD.

`pkg_comp` `pkg_comp`(8) is shipped with `pkgsrc`.