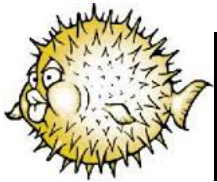
Retrofitting privsep into dpb and pkg_add

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>



September 25, 2016

Retrofitting privsep into dpb and pkg_add

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>

September 25, 2016

Quizz

Which is safer

1. $ cmd
2. # cmd

Quizz

Which is safer

1. $ cmd
2. # cmd

Privsep

- sometimes root is safer
- because it can drop privileges
- how to do it when not root ? pledge(2) is not the answer

Enter DPB

If you missed the previous episodes

- DPB more or less runs `make package` in parallel.
- on multiple hosts
- some steps are separated for better reporting
- some steps (`fetch`) are done by DPB proper (caching etc)

It's all naddy@'s fault

naddy started using fake without root, and it worked surprisingly well

chroot made difficult

also we wanted chroot builds that would work

The problem with sudo (or doas)

SIGINT is not your friend

- add chroot to a build run `sudo chroot /build`
- switch back to normal user `sudo chroot /build sudo -u espie make build`
- try to kill it... doesn't work

Reverse control

Reverse control

- Instead of running as user espie, and using doas to gain privileges we run as root, and drop privileges to user espie
- This solves the SIGINT issue
- Note that most steps don't need root

OO to the rescue

users everywhere

Just qualify code with the user that's supposed to run it

```
$state->{log_user}->run_as(
    sub {
        open my $f, '>', $state->{permanent_log}.'.part' or return;
        for my $p (sort {$a->fullpkgpath cmp $b->fullpkgpath}
            DPB::PkgPath->seen) {
                for my $s (@{$p->{stats}}) {
                        print $f DPB::Serialize::Build->write($s), "\n";
                }
        }
        close $f;
        rename $state->{permanent_log}.'.part', $state->{permanent_log};
    });
```

More OO

```
package DPB::Distfile;
our @ISA = (qw(DPB::UserProxy));
...

$self->run_as(
    sub {
        unlink($name);
    });
```

Of saved uids

Not real privsep

- uses saved uids
- only fully drops when running external commands

Okay perl is a bit weird

```
$( = $gid;
$) = "$gid $gid";
$< = $uid;
$> = $uid;
```

Testing

Just run `id` to make sure.

Trust issues

Who do we trust

- We do assume that dpb code is okay
- We don't trust the ports tree proper
- So each time we run "make" anywhere, we drop privs
- We don't really trust the network
- So each time we run "ftp", we drop privs

Different users

Everybody is different

- One user to fetch stuff
- One user to build stuff
- One user to write logs
- One user that can't do anything

User friendly

Rocket science ?

- We added default users for build (`_pbuild`) and fetch (`_pfetch`) and for dpb proper (`_dpb`) with appropriate defaults.
- `block out quick proto {tcp,udp} from self user _pbuild`
- They get used when you run dpb as root

Granularity

It's actually difficult to know when you've done enough.
We don't have throw-away users yet.

Remaining issues

Installs still need root

Dependency installs want root

Figuring out perms

A bit of a nightmare, who needs to have access to what. Interactive mode in dpb ?

Enter proot

Proot: ports chroot builder

- Preparing chroot environments
- For ports builds on OpenBSD

Help I'm falling asleep

Actually not so much

Why bother

- Existing tools don't match the needs
- It has to be real fast
- It must be damn-fool proof

Preparation preparation preparation

Copy what exactly

- Already have tools (locatedb) that tell us what comprises the base system, so we can copy from it.
- Alternately, start from a snapshot. Also have tools for that.
- Not even close to everything: forego manpages and X server.

One copy

How to do copies

- Speed: do not copy if it didn't change.
- Use hardlinks when we can. Cool and fast cloning of existing chroot

What about the rest

Not enough for a functional system

- you need files for the network
- and ldconfig
- and also devices

ttyname bug

Horrible code

```
static int
oldttyname(struct stat *sb, char *buf, size_t len)
{
        struct dirent *dirp;
        DIR *dp;
        struct stat dsb;

        if ((dp = opendir(_PATH_DEV)) == NULL)
                return (errno);

        while ((dirp = readdir(dp))) {
                if (dirp->d_fileno != sb->st_ino)
                        continue;
                if (dirp->d_namlen > len - sizeof(_PATH_DEV)) {
                        (void)closedir(dp);
                        return (ERANGE);
                }
                memcpy(buf + sizeof(_PATH_DEV) - 1, dirp->d_name,
                    dirp->d_namlen + 1);
                if (stat(buf, &dsb) || sb->st_dev != dsb.st_dev ||
                    sb->st_ino != dsb.st_ino)
                        continue;
                (void)closedir(dp);
                return (0);
        }
        (void)closedir(dp);
        return (ENOTTY);
}
```

ttyname bug 2

practice makes perfect

```
static int
oldttyname(struct stat *sb, char *buf, size_t len)
{
        struct dirent *dirp;
        DIR *dp;
        struct stat dsb;

        if ((dp = opendir(_PATH_DEV)) == NULL)
                return (errno);

        while ((dirp = readdir(dp))) {
                if (dirp->d_type != DT_CHR && dirp->d_type != DT_UNKNOWN)
                        continue;
                if (fstatat(dirfd(dp), dirp->d_name, &dsb, AT_SYMLINK_NOFOLLOW)
                    || !S_ISCHR(dsb.st_mode) || sb->st_rdev != dsb.st_rdev)
                        continue;
                (void)closedir(dp);
                if (dirp->d_namlen > len - sizeof(_PATH_DEV))
                        return (ERANGE);
                memcpy(buf + sizeof(_PATH_DEV) - 1, dirp->d_name,
                    dirp->d_namlen + 1);
                return (0);
        }
        (void)closedir(dp);
        return (ENOTTY);
}
```

ttyname bug 3

Fixes everywhere

- database makes things okay
- so run database
- AND also fix the code!

Design notes

Must be tweakable

- As a default, we remove unknown stuff
- Never under other mount points

Action man

- set of actions, some are default
- some can be added
- ...or removed
- everything needed, writes mk.conf

One size fits all ?

Not really

- ports clusters vary immensely
- because of architectures
- and needs !
- still require 50G+ for distfiles, 50G+ for packages
- takes one day for fast architectures

One size fits all ?

Not really

- ports clusters vary immensely
- because of architectures
- and needs !
- still require 50G+ for distfiles, 50G+ for packages
- takes one day for fast architectures
- I even wrote a manpage for those choices

Turning our eyes to pkg_add

let's do ftp

We simply have a `_pkgfetch` user.

with a clean environment

(now that's the fun part)

how to do env

```
# create sanitized env for ftp
my %newenv = (
        HOME => '/var/empty',
        USER => $user,
        LOGNAME => $user,
        SHELL => '/bin/sh',
        LC_ALL => 'C', # especially, laundry error messages
        PATH => '/bin:/usr/bin'
    );

# copy selected stuff;
for my $k (qw(
    TERM
    FTPMODE
    ...
    FTP_PROXY
    HTTPS_PROXY
    HTTP_PROXY
    NO_PROXY)) {
        if (exists $ENV{$k}) {
                $newenv{$k} = $ENV{$k};
        }
}
# don't forget to swap!
%ENV = %newenv;
```

The trust model of pkg_add

- get stuff from the internet
- unpack it
- check the signature
- install it

extending signify

pulling signatures outside

- We stuff the signature inside the gzip comment
- Signify now has careful code that parses gzip headers
- This is not gzsig
- Chunks of the compressed data are hashed with SHA512/256

not quite finished

almost there

- need to update the signing machines
- new model is stricter `PKG_TRUSTED_PATH`
- `pkg_add` does abysmal reports

In the meantime

The base system installer

- is now privsep'd, thx to rpe@ and halex@
- I strongly suspect rpe@, who's also active in ports, saw the pkg_add work and decided to do the same.

The future

Individual chroot

- One per port, just requires knowing distfiles and packages we need
- hence the hardlinks

The future 2

Security model

- do not need root in the chroot
- make directories writable

Questions !!!