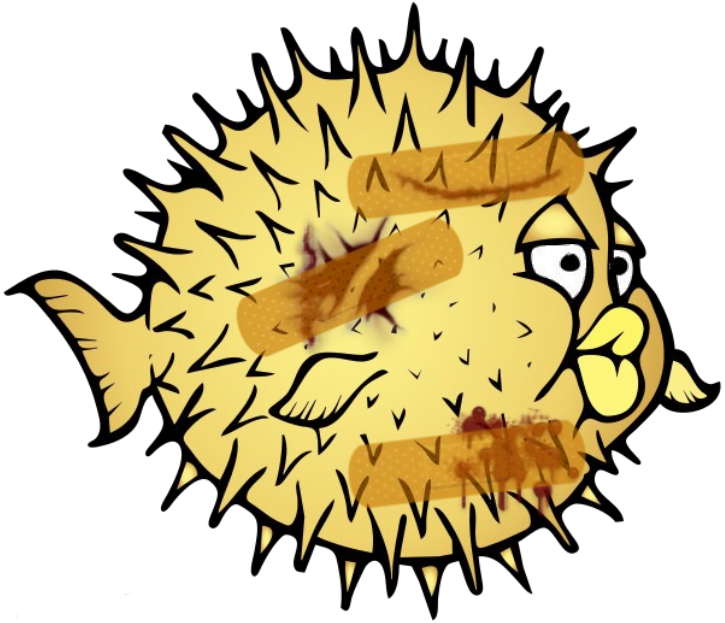
syspatch(8)

The Boring Healing Potion



BSDCan 2019

Antoine Jacoutot
ajacoutot@openbsd.org
@ajacoutot

whoami(1)

- Antoine Jacoutot
- For fun
  - OpenBSD developer since 2006
  - ajacoutot@ aka aja@
  - syspatch, sysmerge, rcctl, rc.d, stuff, more stuff, other stuff...
  - >400 ports
  - ftp.fr.openbsd.org
  - GNOME Foundation member and committer
- For work
  - Engineering manager of operations at:

ManoMano

Agenda

- Introduction
- History
- syspatch(8) usage
- Overview of the errata and build processes
- Challenges building syspatches
- Demo
- Future

syspatch: a tool and a build framework

- */usr/sbin/syspatch* = utility to work with binary patches
  - ksh(1) script (237 LoC as of today)
- */usr/src/distrib/syspatch/* = framework for building binary patches
  - BSD Makefiles

The term “binary patch” is in fact inaccurate: in reality we provide signed tarballs containing updated binaries (no binary diffs)

The Dark Ages

- 2 different solutions, for both src (base system) and xenocara (X.org)
  - Official release (OPENBSD_6_2_BASE) + patch
    - Manually download and verify the patch file using signify(1)
    - cd */usr/src/*... && patch ... && make obj && make && make install
    - On all boxen (workaround: DESTDIR=*/tmp/foo* and tar(1) it up)
  - Stable release (OPENBSD_6_2)
    - cvs up, rebuild both kernel and userland then make a release(8)
    - Only way to be fully patched (e.g. libssl and static linking) and allows for easier redistribution (using the built sets)

The Middle Ages

- M:tier
  - stable.mtier.org provides binary updates for packages and base system
  - Unofficial but several OpenBSD developers are part of the team
  - Binary updates for the base system are distributed as packages
    - Works but abuses the pkg tools
    - Tricky to garbage collect patches from previous release
      - Packages checksum mismatch after upgrading to a new release
  - openup(8)
    - ksh(1) wrapper to keep both packages and the base system up-to-date
    - Mitigate the issue with old irrelevant patches

The Golden Age

- syspatch landed on September 2016
  - g2k16 hackathon in Cambridge, UK
  - Joint work with Robert Nagy (robert@)
  - OpenBSD 6.1 was the first supported release
- Only makes sense on official releases+kernel, not -current nor -stable
- As of today, available for amd64, arm64 and i386
- Supported releases: n & n-1
- Security and reliability maintenance tool, i.e not for release upgrade
- All-or-nothing solution
  - Cumulative: cannot choose which patch to install (either all or none)
    - But step-by-step or full rollback is possible
- KISS: orchestrate available tools

What are others doing?

- Linux: everything is a package
  - dnf upgrade, apt upgrade...
  - Rollbacks can be tricky
  - Can also be used to upgrade to a newer release
- FreeBSD: freebsd-update(8)
  - Real binary patch solution (bsdiff(1)) with rollback feature
  - Can also be used to upgrade to a newer release (but slow)
  - Too far from our needs: we wanted to match the installer (signed tgz)
  - Maybe Package Base soon?
- NetBSD, DragonflyBSD: source only (AFAIK)

System Manager's Manual SYSPATCH(8)

NAME

syspatch – manage base system binary patches

SYNOPSIS

syspatch [-c | -l | -R | -r]

DESCRIPTION

syspatch is a utility to fetch, verify, install and revert OpenBSD binary Patches.

When run without any options, syspatch will apply all missing patches, creating a rollback tarball containing the files it is about to replace, then extracting and installing all files contained in the syspatch tarball. Patches are cumulative and as such it is not possible to install only a subset of them.

The unpriv() function

- su(1) to a non privileged user (*_syspatch:_syspatch*)
  - Matches what the OpenBSD installer and sysupgrade(8) do
- `-f' to create an empty file beforehand using touch(1), chown(1) and chmod(1) so that the unprivileged user can write into it
- Network access and cryptographic verification (signatures) are always done through unpriv()

```
unpriv -f "SHA256.sig" ftp -MVo "SHA256.sig" "https://<...>/SHA256.sig"
```

Check for available patches (cron(1) friendly)

- `syspatch -c`
  - Fetch SHA256.sig (which contains the patches list) using unprivileged ftp(1)
    - Verify SHA256.sig using unprivileged signify(1)
  - Parse and compare SHA256.sig against installed patches
    - If a new patch is available that installs only new files, skip it because it means we don't have the corresponding set installed (typical example: Xorg)
  - Display missing patch(es)

Automatically run by rc(8) after installing or upgrading a system (displayed on console output (`dmesg -s') and mail to root)

List installed patches

- `syspatch -l`
  - Check (specially named) subdirectories under */var/syspatch/* for any rollback tarball
    - rollback is available = patch is installed
    - */var/syspatch* contains the rollback tarball and the signed patch(1) file
  - Display installed patches

Apply all patches

- `syspatch`
  - Checks
    - Remove non matching release */var/syspatch/* content
    - Check for available patches (`-c')
  - Loop
    - Unpriv download and verify (signify(1)) the first available patch
    - Few checks: available space, local RW filesystem...
    - Create a rollback tarball
    - Safe install(1) files (`-S', noop nowadays as its the default behavior)
      - Error = rollback the entire patch
      - If syspatch(8) updated itself, warn the user and end the run
  - End of run
    - If a new kernel was installed: run reorder_kernel (shuffle objects order: KARL)
      - Patches for the kernel only contains modified object files
      - Note: beware of bsd versus bsd.mp

Rollback last installed patch

- `syspatch -r`
  - Same set of checks as when installing patches (size, RO mount...)
  - Extract the most recent rollback tarball: */var/syspatch/${SYSPATCH}/rollback.tgz*
    - Only the last installed patch can be removed, that's by design
      - Again, patches are cumulative
  - Remove the */var/syspatch/${SYSPATCH}/* directory
  - Relink the kernel if needed

Rollback all patches

- `syspatch -R`
  - Run the rollback_patch() function in a loop
    - Similar as looping around `*syspatch -r*' (same steps)
  - Stop at the first error

Building patches: errata release process

- Security or reliability issue is discovered
- Developers knowledgeable on the subject are warned
- A fix gets created, reviewed and committed to -current
- A backport is committed to -stable, the initial errata patch is created
- Patch file is reviewed by developers then deraadt@ signs it
- robert@ and tb@ build the syspatch using the signed patch file
- The syspatch gets tested by developers
- deraadt@ makes the errata signed file and syspatch available on ftp.openbsd.org (and mirrors takes the load)
- www changes are committed and announcement is sent to the lists

Building patches: overview

- Patches are built on net-less machines
- A complete build for each patch
  - Patch type: kernel or full release or full xenocara
  - Installed on the build machine
  - Installed into a fakeroot (DESTDIR, release(8))
    - All fakeroot builds are kept
      - Patches are made by comparing fakeroot with fakeroot-1
      - The initial (unpatched) fakeroot is the extracted official release

Everything is privileged separated

Building patches: Makefile.000

```
# Set ERRATA to match the errata on the mirrors
ERRATA = 000_dummy

# Set BUILD to the type of the errata so that it can be built properly
#       src - run make targets in the specified subdir(s) for base
#       kernel - build both GENERIC and GENERIC.MP
#       xenocara - run make targets in the specified subdir(s) for xenocara
# BUILD = ??? (defaults to src)

.include <bsd.syspatch.mk>
```

Building patches: bsd.syspatch.mk

1. Fetch and verify errata patch file with signify(1) then apply it
2. Create a fakeroot directory within a noperm mount
   a. non-root users can create files owned by anyone, etc.
3. Do a full release (or xenocara or kernel only) build
   a. Using shared object order from the previous build
      i. EXCLUDE_REGEX for readelf(1) `-Ws' in bsd.lib.mk (e.g. emultls.c)
   b. Needed for static linking (e.g. isakmpd(8) and libcrypto)
4. Diff the previous build against the current one: diff.sh
   a. readlink(1) is run on differing files to ensure that only the resolved path is included
   b. Sanity check the list of artifacts to be included (false positives)
5. Create the syspatch tarball (using a .plist file created by diff.sh)
   a. Kernel syspatch only contains differing object files (relinked when applied)

= manual process

Building patches: deterministic builds

- ar(1)
  - Implementation of the **D** flag (BFD_DETERMINISTIC) to prevent some unneeded randomness in archives
    - st_uid, st_gid, st_mtime = 0
    - mode = 644
  - Some part of the build system, src and xenocara required easy modifications to take advantage of this flag
- Static archives timestamp
  - Different for each build
  - cmp(1) is used for comparison in diff.sh
    - Use 34 byte offsets to skip the timestamp index

Building patches: deterministic builds (cont.)

- Shared objects
  - Random objects link order (`sort -R') = different .so for each build
  - The syspatch build framework uses readelf(1) on the shared objects from the previous random build to match their link order: this ends up with the exact same .so file if there was no actual code change
  - gcc -> Clang move
    - Clang's assembler does not properly set .file directives: when compiling assembly files (s|S), the FILE symbol was missing from the object which prevented readelf(1) to see them and get us the link order
      - https://bugs.llvm.org/show_bug.cgi?id=34019
    - Sometimes the location list section (.debug_loc) changes in shared libraries (base system is built using `-g' to add debugging information)
      - Unwanted files end up in the syspatch .plist (needs manual editing)
      - Got fixed upstream and merge into OpenBSD

Building patches: deterministic builds (cont.)

- perl(1)
  - Configure test had several problems that resulted in excess bytes not getting zeroed out causing random contents in $Config{longdblinfbytes}; fixed since
  - Manual pages created with Pod::Man(3p) include the build date
    - e.g. .TH POD2HTML 1 "2018-01-18" "perl v5.24.3"
    - diff.s h uses sed(1) to remove the date from comparison
- __TIME__ and __DATE__
  - Used by some software version info to display the build time and date
  - e.g. fvwm(1)
  - Easy fix: remove it from the code (it's mostly a useless feature)
- ld.so(1)
  - ld.so was missing the FILE symbol (which contains the objects build order)
  - Remove `-x' from `ld(1)' in ld.so Makefile for syspatch builds only (delete all local symbols)

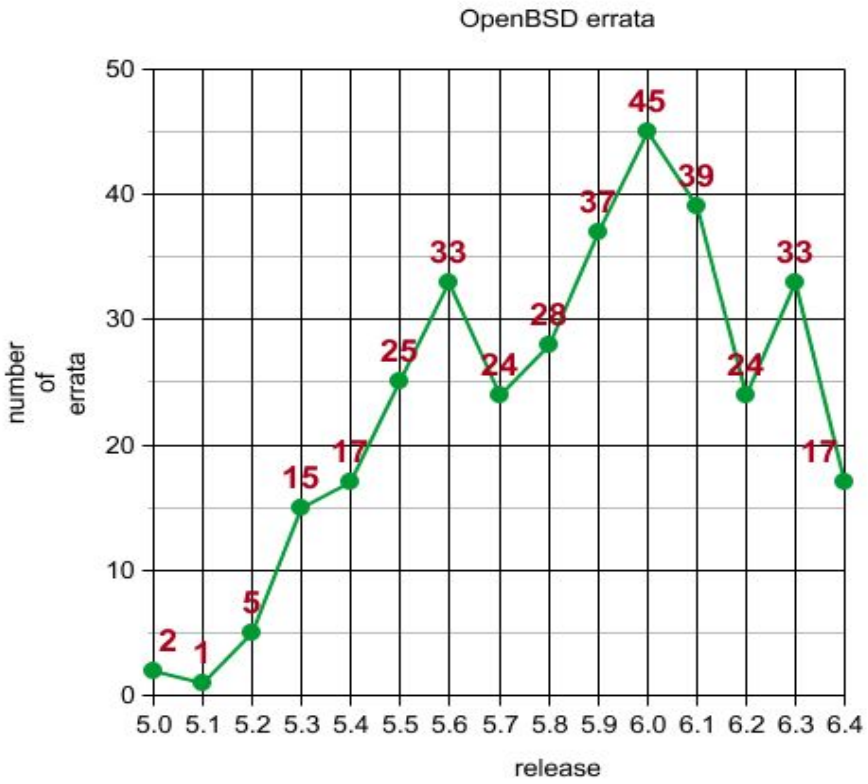
Building patches: deterministic builds (cont.)

- Patches aren’t built on the release build machines
  - That triggered an issue in our release build process because:
    - No automated garbage collection of old files (e.g. old headers)
    - May introduce build differences (e.g. configure could pick something up)
    - Only relevant for the first syspatch (official release against first build comparison)
  - The syspatch build machines helped made our release builds cleaner

Demo

```
$ syspatch -h
/usr/sbin/syspatch[297]: -h: unknown option
usage: syspatch [-c | -l | -R | -r]
```

Some statistics



- 6.0 private preview
- 6.1 public preview
- 6.2+ “trust me, I’m an engineer”

What can we learn from this?

The future

- Support more architectures
- On rollback: remove files that were installed by a syspatch but didn't exist before
- One */bsd* to rule them all
- Pave the road for stable packages
  - Challenge != technical but needs infrastructure and manpower
- sysupgrade(8) is coming
  - sysupgrade -> sysmerge -> syspatch

Questions?

syspatch(1) initial design and implementation was done during a hackathon, your donations are put to good use! ;-)

Thank you!

BSDCan 2019

Antoine Jacoutot
ajacoutot@openbsd.org
@ajacoutot