

# Reproducible Builds, a year later

Lunar <lunar@debian.org>

2014-08-26 DebConf14



# What are reproducible builds?

“reproducible” builds enable anyone to reproduce the exact same binary packages from a given source



# Why?


- Prevent targeted attacks
- Debugging: ensure known source; create missing debug symbols
- Ensure packages can be built from source
- Help building Multi-Arch: same packages
- Similar .deb: deduplication, small deltas
- Different build profiles, same common packages



# How did this start?

https://blog.torproject.org/blog/deterministic-builds-part-one-cyberwar-and-global-compromise

Startpage



HOME ARCHIVES ABOUT TOR

## Deterministic Builds Part One: Cyberwar and Global Compromise

View Edit

Posted August 20th, 2013 by mkeperry in [cyberpeace](#), [dangerous toys](#), [decentralization](#), [deterministic builds](#), [ghisla](#), [lack of foresight](#), [National Insecurity Agency](#), [security](#)

I've spent the past few months developing a new build system for the [3.0 series](#) of the Tor Browser Bundle that produces what are called "deterministic builds" -- packages which are byte-for-byte identical no matter who actually builds them, or what hardware they use. This effort was extraordinarily involved, consuming all of my development time for over two months (including several nights and weekends), babysitting builds and fixing differences and issues that arose.

When describing my recent efforts to others, by far the two most common questions I've heard are "Why did you do that?" and "How did you do that?". I've decided to answer each question at length in a separate blog post. This blog post attempts to answer the first question: "Why would anyone want a deterministic build process?"

The short answer is: to protect against targeted attacks. Current popular software development practices simply cannot survive targeted attacks of the scale and scope that we are seeing today. In fact, I believe we're just about to witness the first examples of large scale "watering hole" attacks. This would be malware that attacks the software development and build processes themselves to distribute copies of itself to tens or even hundreds of millions of machines in a single, officially signed, instantaneous update. Deterministic, distributed builds are perhaps the only way we can reliably prevent these types of targeted attacks in the face of the endless stockpiling of weaponized exploits and other "cyberweapons".

- Add a New Blog Post
- Manage Blog
- Admin Comments
- Manage Users
- Add an Event
- Manage Events
- Manage Forums

### Search

# Nothing new

From: Martin Uecker <muecker@gmx.de>

Cc: debian-devel@lists.debian.org

Date: Sun, 23 Sep 2007 23:32:59 +0200

*I think it would be really cool if the Debian policy required that packages could be rebuild bit-identical from source. At the moment, it is impossible to independly verify the integricity of binary packages.*

<https://lists.debian.org/debian-devel/2007/09/msg00746.html>



# Although, reactions were not enthusiastic

From: Neil Williams <codehelp@debian.org>

To: debian-devel@lists.debian.org

Date: Mon, 24 Sep 2007 07:22:30 +0100

- > *Then third parties can recreate the binaries*
- > *and publish recreated hashes.*

*Why? I see no benefit.*

<https://lists.debian.org/debian-devel/2007/09/msg00747.html>



# Although, reactions were not enthusiastic

From: Manoj Srivastava <srivasta@debian.org>  
To: debian-devel@lists.debian.org  
Date: Sun, 23 Sep 2007 23:25:16 -0500

*I, for one, think this technically infeasible, but hey, I'll be happy to be proved wrong.*

<https://lists.debian.org/debian-devel/2007/09/msg00760.html>



# BoF during DebConf13

- Planned at the last minute
- 30 attendees
- Kicked off

[wiki.debian.org/ReproducibleBuilds](http://wiki.debian.org/ReproducibleBuilds)





# How?

- Record the build environment
- Reproduce the build environment
- Eliminate unneeded variations



# Record the build environment

Record which versions of the build dependencies (and their dependencies) are installed.



# Reproduce the build environment

[snapshot.debian.org](http://snapshot.debian.org)



# Source of variations

- Timestamps
- Build paths
- File order
- Locale
- ...



# Timestamps

gzip stores a timestamp.

```
$ file README.txt.gz
README.txt.gz: gzip compressed data, was "README.txt", from Unix,
last modified: Mon Mar  5 00:05:49 2012, max compression
```



# Timestamps

ar, tar, zip, jar... store timestamps.

```
$ tar ztvf copyright-format.xml.tar.gz
-rw-r--r-- pbuilder/pbuilder  473 2012-03-05 00:02 Makefile
-rw-r--r-- pbuilder/pbuilder 56918 2012-03-05 00:05 copyright-format-1.0.html
-rw-r--r-- pbuilder/pbuilder 37218 2012-03-05 00:05 copyright-format-1.0.txt
-rw-r--r-- pbuilder/pbuilder 10007 2012-03-05 00:05 copyright-format-1.0.txt.gz
-rw-r--r-- pbuilder/pbuilder 53917 2012-03-05 00:02 copyright-format-1.0.xml
-rw-r--r-- pbuilder/pbuilder   808 2012-03-05 00:02 html.dsl
-rw-r--r-- pbuilder/pbuilder   97 2012-03-05 00:05 version.xml
```



# Timestamps

## javadoc writes timestamps:

```
$ head -n 5 /usr/share/doc/libjaxe-java-doc/api/serialized-form.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<!-- NewPage -->
<html lang="en">
<head>
<!-- Generated by javadoc (version 1.6.0_27) on Sat Jul 13 17:27:51 UTC 2013 -->
```



# Build paths

## Build path is embedded in debug symbols:

```
$ readelf -w /usr/lib/debug/usr/bin/pidgin | grep '/tmp/build' | head -n 4
<11>    DW_AT_name      : /tmp/build/pidgin-2.10.6/./pidgin/pidginstock.c
<15>    DW_AT_comp_dir  : /tmp/build/pidgin-2.10.6/build/pidgin
<402d> DW_AT_name      : /tmp/build/pidgin-2.10.6/./pidgin/gtkaccount.c
<4031> DW_AT_comp_dir  : /tmp/build/pidgin-2.10.6/build/pidgin
```





# File order

`readdir()` returns file in the order of the file system.



# Locale

Behaviour can change depending on configured locale:

```
$ printf 'a\nâ\nb\n' | LC_ALL=C.UTF-8 sort
```

```
a  
b  
â
```

```
$ printf 'a\nâ\nb\n' | LC_ALL=fr_FR.UTF-8 sort
```

```
a  
â  
b
```



# Misc.

- Hostname
- Uname output
- Username



# Cheat

- Use a VM: same kernel, same user, same build path
- `libfaketime`



# The hard path

- Configure the toolchain:  
`binutils --enable-deterministic-archives`
- Add missing options:  
`javadoc --no-timestamps`
- Patch build systems:  
`gzip -n`



# Experiment

- Build and rebuild of many source packages
- Using EC2 VM instances from Amazon Web Services
- Many thanks David Suárez!



# Experiment

- Build packages twice
- Setup clean chroot, unpack source code, install build-deps, build
- And again...
- Pass the timestamp of the first build to dpkg through environment variable



# Experiment

Variations in this context:

- Time
- Build path

No changes in hostname, username, uname, file order, locale...





# Experiment

Modified packages for the January 2014 experiment:

- dpkg: use single timestamp in the archives
- dpkg: re-use timestamp from environment if given
- dpkg: stable file order in the archives
- debhelper: dh\_strip calls debugedit
- dpkg: pass `-fno-merge-debug-strings` through `dpkg-buildflags`
- binutils: built with `--enable-deterministic-archives`



# Experiment

- Upon 5151 source packages
- 3196 produced identical binary packages



# Experiment

62%

Waow.



# Already reproducible

source name	popcon insts
-----	-----
findutils	164641
wget	164512
klibc	163312
busybox	161494
installation-report	157494
laptop-detect	157352
python-support	155075
netkit-ftp	145548



# Failures in the remaining packages

```
1017 build-id-mismatch
295 unknown
108 jar-file
106 haskell-prof
103 haskell-dev
101 php-registry
101 html-mismatch
63 same-depends-different-order
62 r-rds
52 gzip-timestamp
46 kde-doc-index
```



# Failures in the remaining packages

45 mono  
35 specific  
33 docbook-to-man-timestamp  
23 do-not-use-dpkg-buildflags  
21 debugedit-not-run-or-failed  
16 puredata  
13 perl-manpage  
11 rdoc-timestamp  
10 zip-file  
8 ocaml-md5sums  
7 fonts  
7 erlang



# Further research

Still no good solution for the build ID issue.



# Further research

- How about deciding on a canonical build path?  
`~/usr/src/debian/hello-2.9``
- `proot` can fake the current directory like `fakeroot` fakes `uid`.
- `gdb` would be able to easily look up source code.

Thanks to Stéphane Glondu for working the idea.





# Further research

Should `dpkg-buildpackage export GZIP=-n`?



# More experiments

- New `dpkg-buildpackage` patch that will call `proot`.
- Unfortunately, David Suárez was too busy this time to help with archive-wide experimentation.



# Other distributions

- Fedora  
<http://securityblog.redhat.com/2013/09/18/reproducible-builds-for-fedora/>
- OpenSUSE build-compare  
<https://build.opensuse.org/package/show/openSUSE:Factory/build-compare>
- NixOS  
<http://lists.science.uu.nl/pipermail/nix-dev/2013-June/011357.html>



# Want to help?

Triage:

- Let's make a new archive-wide rebuild and sort the result.



# Want to help?

Specify:

- Think about the best way to record the environment.



# Want to help?

## Code:

- Add “no timestamps” option to jar, javadoc, epydoc...
- Write a script to rebuild a package from a .changes file and a recorded environment.



# Want to help?

## Project management:

- Coordinate the baby steps needed to move this forward.



# Want to help?

## Stay in touch:

- [Subscribe to the ReproducibleBuilds wiki page.](#)
- [Subscribe to the reproducible-builds@l.a.d.o mailing list.](mailto:reproducible-builds@l.a.d.o)





BoF

BoF to discuss technical solutions at 19:00 in room 329



Questions? Comments?

?

[wiki.debian.org/ReproducibleBuilds](http://wiki.debian.org/ReproducibleBuilds)

