Documenting and Fixing Non-Reproducible Builds due to Configuration Options

FOSDEM'24

RANDRIANAINA Georges Aaron

georges-aaron.randrianaina@irisa.fr

Université de Rennes, Inria/IRISA UMR 6074, DiverSE team, France

Reproducible Builds Definition

“The build process of a software product is **reproducible** if, after designating a specific version of its source code and all of its build dependencies, **every build produces bit-for-bit identical artifacts**, **no matter the environment** in which the build is performed.”

C. Lamb and S. Zacchiroli, "*Reproducible Builds: Increasing the Integrity of Software Supply Chains*," in IEEE Software, 2022

Reproducible Builds Guidelines

- Achieve deterministic builds
  - Variations in the build environment
  - `SOURCE_DATE_EPOCH`
  - Deterministic build systems
  - Volatile inputs can disappear
  - Stable order for inputs
  - Value initialization
  - Version information
  - Timestamps
  - Timezones
  - Locales
  - Archive metadata
  - Stable order for outputs
  - Randomness
  - Build path
  - System images
  - JVM
- Define a build environment
  - What's in a build environment?
  - Recording the build environment
  - Definition strategies
  - Proprietary operating systems
- Distribute the environment
  - Building from source
  - Virtual machine drivers
  - Formal definition
- ...

Reproducible Builds

https://reproducible-builds.org/docs

Example: Linux

```
$ cd linux
$ make tinyconfig
$ make -j16
$ mv vmlinux /tmp/vmlinux1
$ git clean -dfx
$ make tinyconfig
$ make -j16
$ diffoscope /tmp/vmlinux1 vmlinux
```

Example: Linux

```
$ cd linux
$ make tinyconfig
$ make -j16
$ mv vmlinux /tmp/vmlinux1
$ git clean -dfx
$ make tinyconfig
$ make -j16
$ diffoscope /tmp/vmlinux1 vmlinux
  0xc10ce260 00000023 31204d6f 6e204f63 74203920 ...#1 Mon Oct 9
- 0xc10ce270 31343a32 373a3139 20434553 54203230 14:27:19 CEST 20
+ 0xc10ce270 31343a32 373a3336 20434553 54203230 14:27:36 CEST 20
  0xc10ce280 32330000 00000000 00000000 00000000 23..............
```

Example: Linux

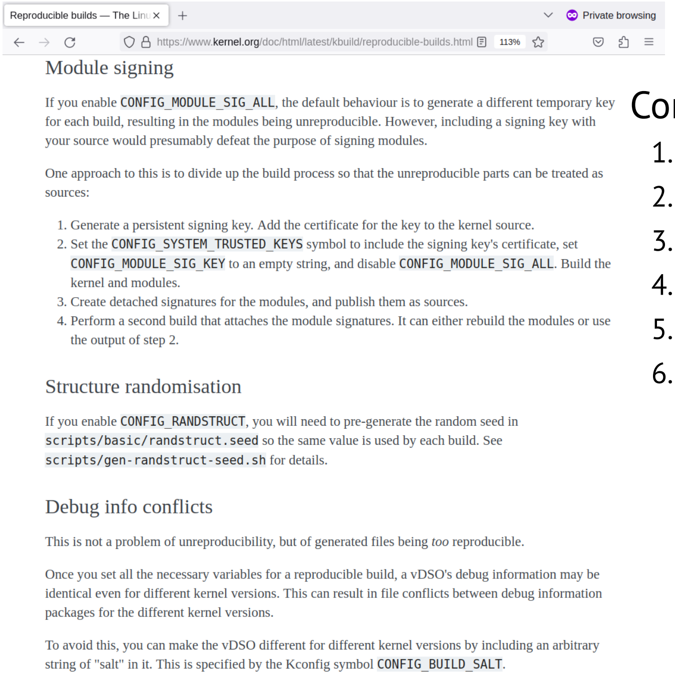
```
$ cd linux
$ make tinyconfig
$ make -j16
$ mv vmlinux /tmp/vmlinux1
$ git clean -dfx
$ make tinyconfig
$ make -j16
$ diffoscope /tmp/vmlinux1 vmlinux
  0xc10ce260 00000023 31204d6f 6e204f63 74203920 ...#1 Mon Oct 9
- 0xc10ce270 31343a32 373a3139 20434553 54203230 14:27:19 CEST 20
+ 0xc10ce270 31343a32 373a3336 20434553 54203230 14:27:36 CEST 20
  0xc10ce280 32330000 00000000 00000000 00000000 23..............
```

- Can be solved by setting Kbuild's variable `KBUILD_BUILD_TIMESTAMP` to a fixed string (e.g., ''`Sun Jan 1 01:00:00 UTC 2023`'')

Example: Linux

```
$ cd linux
$ make tinyconfig
$ make -j16
$ mv vmlinux /tmp/vmlinux1
$ git clean -dfx
$ make tinyconfig
$ make -j16
$ diffoscope /tmp/vmlinux1 vmlinux
  0xc10ce260 00000023 31204d6f 6e204f63 74203920 ...#1 Mon Oct 9
- 0xc10ce270 31343a32 373a3139 20434553 54203230 14:27:19 CEST 20
+ 0xc10ce270 31343a32 373a3336 20434553 54203230 14:27:36 CEST 20
  0xc10ce280 32330000 00000000 00000000 00000000 23..............
```

- Can be solved by setting Kbuild's variable `KBUILD_BUILD_TIMESTAMP` to a fixed string (e.g., ''`Sun Jan 1 01:00:00 UTC 2023`'')
- Ok, but how about `defconfig`, `allyesconfig`, `allmodconfig`, `randconfig`, ...?!


Reproducible builds — The Linu

https://www.kernel.org/doc/html/latest/kbuild/reproducible-builds.html

Module signing

If you enable `CONFIG_MODULE_SIG_ALL`, the default behaviour is to generate a different temporary key for each build, resulting in the modules being unreproducible. However, including a signing key with your source would presumably defeat the purpose of signing modules.

One approach to this is to divide up the build process so that the unreproducible parts can be treated as sources:

1. Generate a persistent signing key. Add the certificate for the key to the kernel source.
2. Set the `CONFIG_SYSTEM_TRUSTED_KEYS` symbol to include the signing key's certificate, set `CONFIG_MODULE_SIG_KEY` to an empty string, and disable `CONFIG_MODULE_SIG_ALL`. Build the kernel and modules.
3. Create detached signatures for the modules, and publish them as sources.
4. Perform a second build that attaches the module signatures. It can either rebuild the modules or use the output of step 2.

Structure randomisation

If you enable `CONFIG_RANDSTRUCT`, you will need to pre-generate the random seed in `scripts/basic/randstruct.seed` so the same value is used by each build. See `scripts/gen-randstruct-seed.sh` for details.

Debug info conflicts

This is not a problem of unreproducibility, but of generated files being *too* reproducible.

Once you set all the necessary variables for a reproducible build, a vDSO's debug information may be identical even for different kernel versions. This can result in file conflicts between debug information packages for the different kernel versions.

To avoid this, you can make the vDSO different for different kernel versions by including an arbitrary string of "salt" in it. This is specified by the Kconfig symbol `CONFIG_BUILD_SALT`.

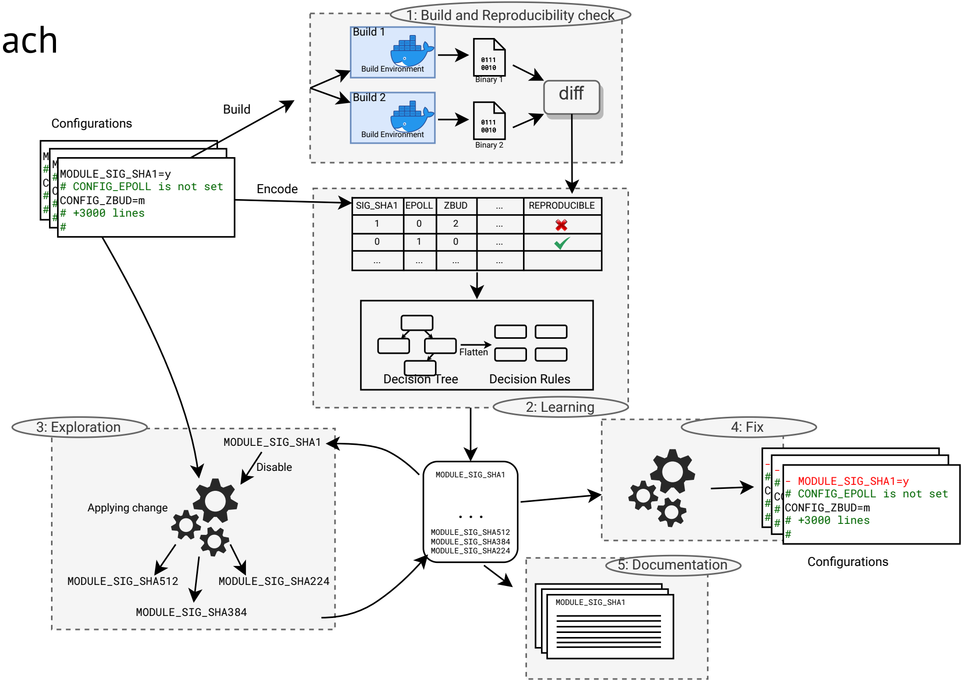
Configuration options

1. `CONFIG_IKHEADERS`
2. `CONFIG_MODULE_SIG_ALL`
3. `CONFIG_SYSTEM_TRUSTED_KEYS`
4. `CONFIG_MODULE_SIG_KEY`
5. `CONFIG_RANDSTRUCT`
6. `CONFIG_BUILD_SALT`

That's all?

The Linux kernel has 19K+ configuration options

Approach



```
MODULE_SIG_SHA1=y
# CONFIG_EPOLL is not set
CONFIG_ZBUD=m
# +3000 lines
#
```

SIG_SHA1	EPOLL	ZBUD	...	REPRODUCIBLE
1	0	2	...	✗
0	1	0	...	✓
...	...	...	...	

```
MODULE_SIG_SHA1
. . .
MODULE_SIG_SHA512
MODULE_SIG_SHA384
MODULE_SIG_SHA224
```

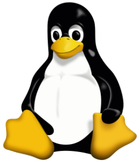
```
- MODULE_SIG_SHA1=y
# CONFIG_EPOLL is not set
CONFIG_ZBUD=m
# +3000 lines
#
```

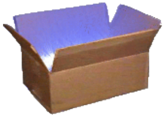
```
MODULE_SIG_SHA1
```

Setup

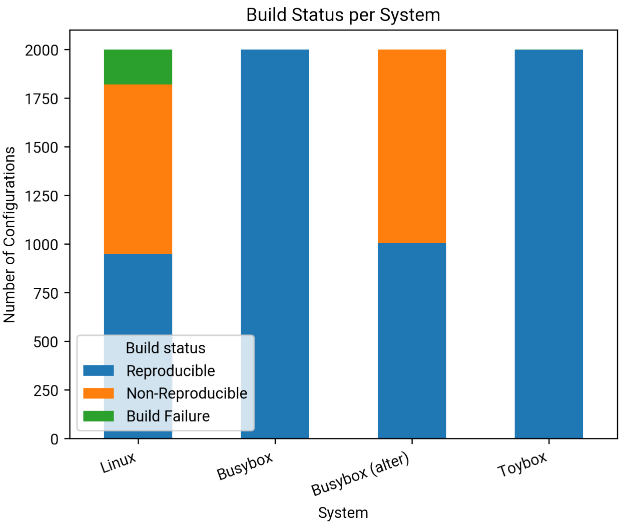
- Configurations (2000 each)
  - Linux: `randconfig` + `X86_64` preset
  - Busybox & Toybox: custom scripts (also using `randconfig`)
- Build environment
  - Docker image derived from Tuxmake (+ few more packages)
  - `KBUILD_BUILD_TIMESTAMP`
  - `KBUILD_BUILD_HOST`
  - `KBUILD_BUILD_USER`
  - `KBUILD_BUILD_VERSION`
  - `KBUILD_NOTIMESTAMP`
- Machine: Debian 12 (Bookworm), AMD Epyc 7532, 512 GB of RAM

System	**Version**	**LoC**	**#Options**
Linux	5.13	21605254	18637
Busybox	1.36.1	220304	1093
Toybox	0.8.5	69355	341



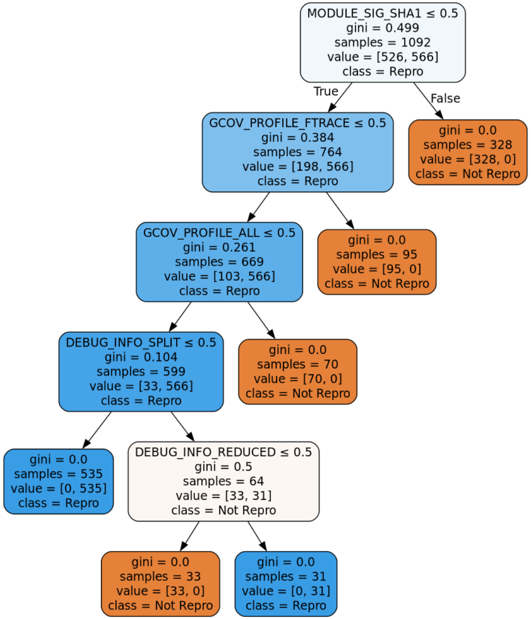


Reproducibility of configurations' build



- *Linux:* **47.45% of non-reproducible builds**, 8.95% of build failure
- *Busybox (alter):* **interactions across layers exist** (e.g., compile-time option `DEBUG` with build path) and may hamper reproducibility – **49.75%** of the builds **non-reproducible**
- *Toybox:* 1 build failure over 2000 builds, **100% reproducible!**

Who is to blame?



1. MODULE_SIG_SHA1
2. GCOV_PROFILE_FTRACE
3. GCOV_PROFILE_ALL
4. DEBUG_INFO_SPLIT
5. DEBUG_INFO_REDUCED

Exploration

- Main idea: identify the same kind of options that have the same effect
- We identify "siblings" of the options
- We leverage Kconfig's naming convention to get the parent of the option.

1. `MODULE_SIG_SHA1`
2. `GCOV_PROFILE_FTRACE`
3. `GCOV_PROFILE_ALL`
4. `DEBUG_INFO_SPLIT`
5. `DEBUG_INFO_REDUCED`

$\longrightarrow$

1. `MODULE_SIG_SHA1`
2. `MODULE_SIG_SHA224`
3. `MODULE_SIG_SHA256`
4. `MODULE_SIG_SHA384`
5. `MODULE_SIG_SHA512`
6. `MODULE_SIG`
7. `GCOV_PROFILE_FTRACE`
8. `GCOV_PROFILE_ALL`
9. `DEBUG_INFO_SPLIT`
10. `DEBUG_INFO_REDUCED`

Fix

- Remove detected options and their dependencies from the configuration
  Dependency computation done with ConfigFix (SAT-based solver)

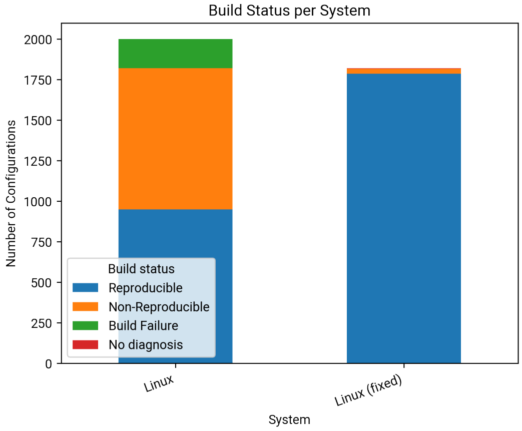
  P. Franz, T. Berger, I. Fayaz, S. Nadi and E. Groshev, "*ConfigFix: Interactive Configuration Conflict Resolution for the Linux Kernel*," ICSE-SEIP 2021
- Build and check for reproducibility

Fix

- Remove detected options and their dependencies from the configuration
  Dependency computation done with ConfigFix (SAT-based solver)

  P. Franz, T. Berger, I. Fayaz, S. Nadi and E. Groshev, "*ConfigFix: Interactive Configuration Conflict Resolution for the Linux Kernel*," ICSE-SEIP 2021

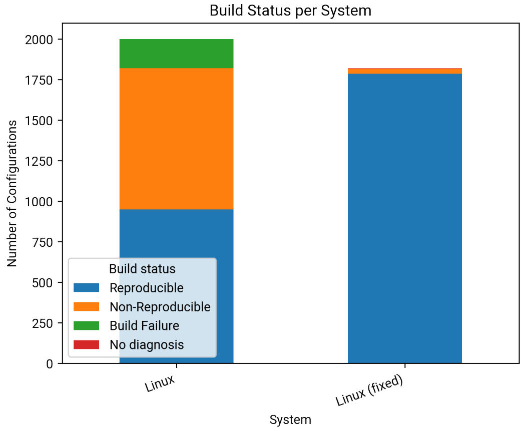
- Build and check for reproducibility



Fix

- Remove detected options and their dependencies from the configuration
  Dependency computation done with ConfigFix (SAT-based solver)

  P. Franz, T. Berger, I. Fayaz, S. Nadi and E. Groshev, "*ConfigFix: Interactive Configuration Conflict Resolution for the Linux Kernel*," ICSE-SEIP 2021
- Build and check for reproducibility



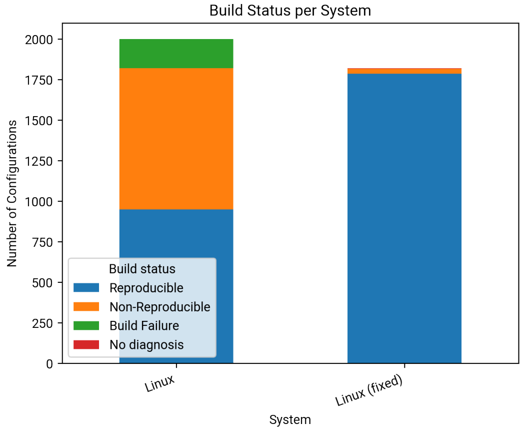
- **96% of non-reproducible builds made reproducible**
- 31 configurations (**3.5%**) still non-reproducible
- 3 configurations (**<0.5%**) for which ConfigFix could not find diagnosis for the change

Fix

- Remove detected options and their dependencies from the configuration
  Dependency computation done with ConfigFix (SAT-based solver)

  P. Franz, T. Berger, I. Fayaz, S. Nadi and E. Groshev, "*ConfigFix: Interactive Configuration Conflict Resolution for the Linux Kernel*," ICSE-SEIP 2021

- Build and check for reproducibility



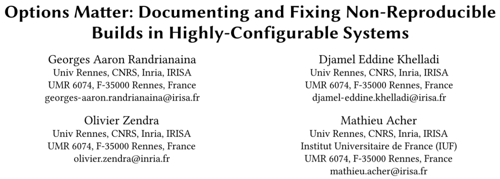
- **96% of non-reproducible builds made reproducible**
- 31 configurations (**3.5%**) still non-reproducible
- 3 configurations (**<0.5%**) for which ConfigFix could not find diagnosis for the change
- From **47.45% to 1.4%** of non-reproducible builds

Summary

- Configuration options have an impact on build reproducibility
- Interactions across variability layers exist and may hamper reproducibility
- We have identified a list of novel configuration options that cause non-reproducibility and that do not appear in the doc
- Removing detected options made 96% of non-reproducible builds reproducible

Summary

- Configuration options have an impact on build reproducibility
- Interactions across variability layers exist and may hamper reproducibility
- We have identified a list of novel configuration options that cause non-reproducibility and that do not appear in the doc
- Removing detected options made 96% of non-reproducible builds reproducible



Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems

Georges Aaron Randrianaina
Univ Rennes, CNRS, Inria, IRISA
UMR 6074, F-35000 Rennes, France
georges-aaron.randrianaina@irisa.fr

Djamel Eddine Khelladi
Univ Rennes, CNRS, Inria, IRISA
UMR 6074, F-35000 Rennes, France
djamel-eddine.khelladi@irisa.fr

Olivier Zendra
Univ Rennes, CNRS, Inria, IRISA
UMR 6074, F-35000 Rennes, France
olivier.zendra@inria.fr

Mathieu Acher
Univ Rennes, CNRS, Inria, IRISA
Institut Universitaire de France (IUF)
UMR 6074, F-35000 Rennes, France
mathieu.acher@irisa.fr

MSR '24

21st INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES

April 15-16, Lisbon, Portugal

Summary

- Configuration options have an impact on build reproducibility
- Interactions across variability layers exist and may hamper reproducibility
- We have identified a list of novel configuration options that cause non-reproducibility and that do not appear in the doc
- Removing detected options made 96% of non-reproducible builds reproducible

Options Matter: Documenting and Fixing Non-Reproducible Builds in Highly-Configurable Systems

Georges Aaron Randrianaina
Univ Rennes, CNRS, Inria, IRISA
UMR 6074, F-35000 Rennes, France
georges-aaron.randrianaina@irisa.fr

Djamel Eddine Khelladi
Univ Rennes, CNRS, Inria, IRISA
UMR 6074, F-35000 Rennes, France
djamel-eddine.khelladi@irisa.fr

Olivier Zendra
Univ Rennes, CNRS, Inria, IRISA
UMR 6074, F-35000 Rennes, France
olivier.zendra@inria.fr

Mathieu Acher
Univ Rennes, CNRS, Inria, IRISA
Institut Universitaire de France (IUF)
UMR 6074, F-35000 Rennes, France
mathieu.acher@irisa.fr

MSR '24

21st INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES

April 15-16, Lisbon, Portugal

Thank You. Questions?