What goes into a binary?

Armijn Hemel
Tjaldur Software Governance Solutions

October 28, 2011

About Armijn

- using Open Source software since 1994
- MSc Computer Science from Utrecht University (The Netherlands)
- core team `gpl-violations.org` since 2005
- ex-board member at NLUUG (`http://www.nluug.nl/`)
- sysadmin, developer and consultant at Loohuis Consulting (2006 - May 2011)
- May 2011 - present: owner Tjaldur Software Governance Solutions

Our research & this talk

We argue:

- not granular enough: for example package information in distributions is too coarse and often incorrect
- static (source code) analysis alone is not enough to determine the right license of a binary
- we all need to do a better job

This is not (necessarily) a pure licensing talk! One of the applications is related to licensing, so focus will be on that.

Also: this is unpublished research (sent to International Conference on Software Engineering 2012), so still rough.

The team

- Eelco Dolstra (Delft University of Technology, the Netherlands)
- Sander van der Burg (Delft University of Technology, the Netherlands)
- Daniel M. German (University of Victoria, Canada)
- Julius Davies (University of Victoria, Canada)
- Armijn Hemel (Tjaldur Software Governance Solutions, the Netherlands)

WANAL

Licensing is hard

Doing correct licensing is not trivial and there are many unclarities:

- Linux kernel
- Amarok
- KDE
- opkg

Example: Linux kernel

GPLv2 licensed, or is it?

> *This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 1*

(source: `drivers/net/cs89x0.h`)

Example: Amarok

Amarok is a popular music player originating from KDE, licensed as GPLv2+ (with KDE exceptions).

For two years some files were GPLv3+ by accident because of a typo.

Example: KDE

What license is this?

> *This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation version 2.0*

(source: `kioslave/metainfo/metainfo.cpp` in `kdelibs`)

It is very likely they actually meant LGPL 2.

In short: many projects have suboptimal, or unclear (or no) license statements. We can do better here!

Example: opkg

opkg is a package manager that is used on embedded Linux distributions.

Question: given a binary of opkg, what license(s) can it be distributed under?

GPLv2? GPLv2+? GPLv3+?

opkg has a COPYING file containing the text of GPLv2.

All source code files in opkg are GPLv2+ **except** libopkg/sha256.c and libopkg/sha256.h which are GPLv3+!

These files are not always included, but they are most of the time. The configure script has a switch:

--enable-sha256 Enable sha256sum check [default=yes]
Correct answer: it depends and more information about the composition of the binary is needed.

Possible solution: static analysis?

Static analysis (source code level) can tell you a lot, but information is vastly incomplete:

- many different types of build systems and scripts
- output from `configure` has huge influence
- environment variables set by scripts or users

Better solution: tracing the build

We can track system calls using `strace` and see which files are used by a build!

- no need to modify existing build tools
- (pretty much) standard package for Linux
- build system agnostic
- no special privileges needed

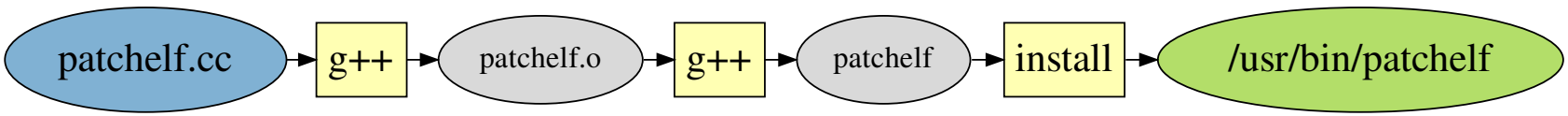
Minor drawbacks:

- a bit verbose
- performance hit

Trace example output

```
open("patchelf.cc", O_RDONLY)
open("/usr/include/c++/4.5.1/string",
  O_RDONLY|O_NOCTTY)
open("elf.h", O_RDONLY)
```

Result: build graph



Pruning the result graph

Only tracing which files are used gives a *conservative* estimate of which files are used.

False positives (files that are opened, but not used for building the actual binary) can be pruned by adding more intelligence.

Applications

Use result set:

- determine license of each file (using Ninka, or FOSSology)
- scan for security bugs
- ...

Early success: FFmpeg

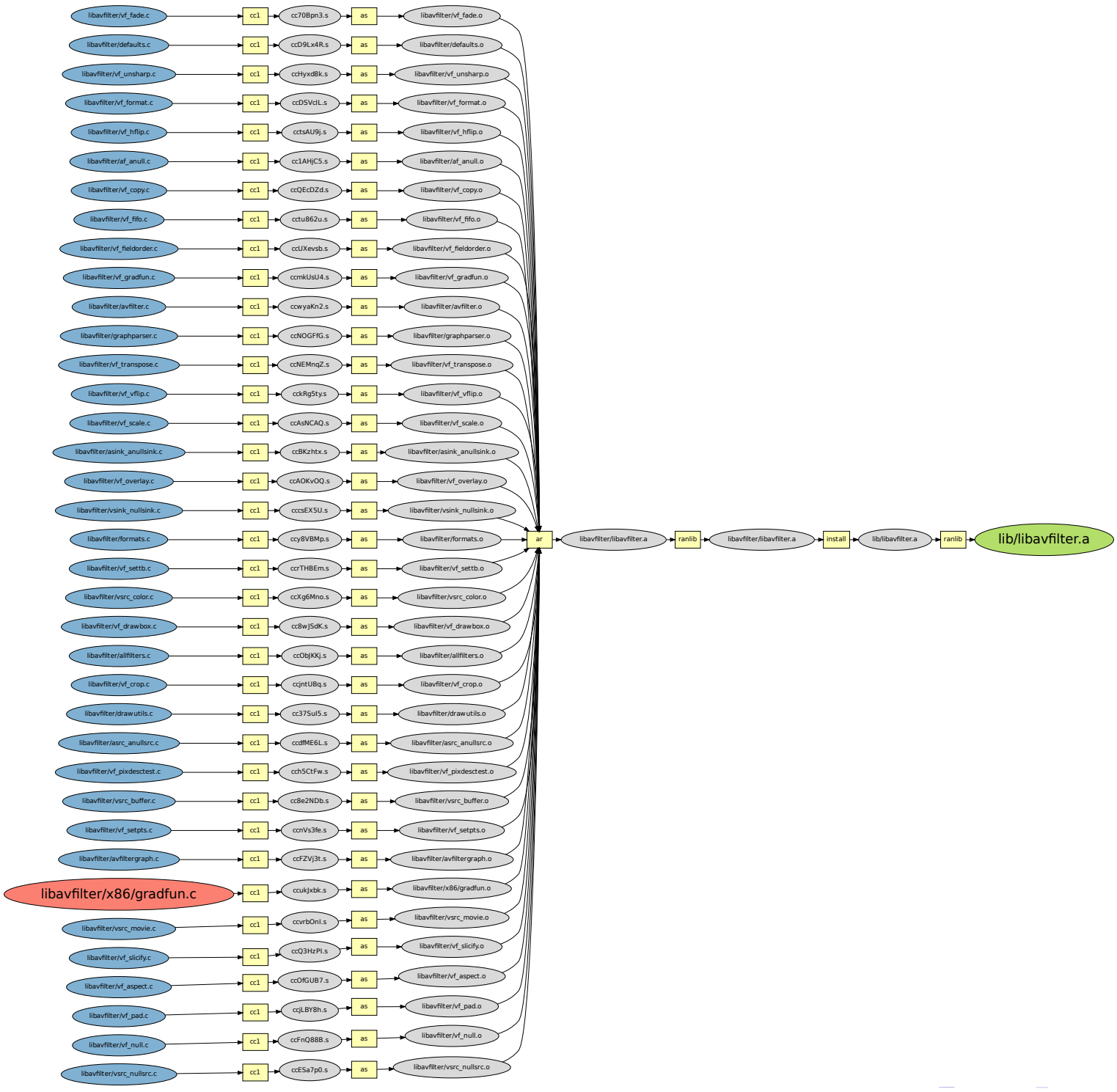
FFmpeg is a mix of GPLv2+ and LGPLv2.1+ licensed code. The configure script has an option to only use the LGPLv2.1+ sources for a build.

With our approach we found that some GPLv2+ code was *always* included in libavfilter.

The offending code was in libavfilter/x86/gradfun.c, licensed under the GPLv2+.

This was not trivial to find out from the FFmpeg build scripts.

FFmpeg fixed it within hours after being informed.


libavfilter/vf_fade.c
cc1
cc70Bpn3.s
as
libavfilter/vf_fade.o
libavfilter/defaults.c
ccD9Lx4R.s
libavfilter/defaults.o
libavfilter/vf_unsharp.c
ccHyxd8k.s
libavfilter/vf_unsharp.o
libavfilter/vf_format.c
ccDSVclL.s
libavfilter/vf_format.o
libavfilter/vf_hflip.c
cctsAU9j.s
libavfilter/vf_hflip.o
libavfilter/af_anull.c
cc1AHjC5.s
libavfilter/af_anull.o
libavfilter/vf_copy.c
ccQEcDZd.s
libavfilter/vf_copy.o
libavfilter/vf_fifo.c
cctu862u.s
libavfilter/vf_fifo.o
libavfilter/vf_fieldorder.c
ccUXevsb.s
libavfilter/vf_fieldorder.o
libavfilter/vf_gradfun.c
ccmkUsU4.s
libavfilter/vf_gradfun.o
libavfilter/avfilter.c
ccwyaKn2.s
libavfilter/avfilter.o
libavfilter/graphparser.c
ccNOGFfG.s
libavfilter/graphparser.o
libavfilter/vf_transpose.c
ccNEMnqZ.s
libavfilter/vf_transpose.o
libavfilter/vf_vflip.c
cckRg5ty.s
libavfilter/vf_vflip.o
libavfilter/vf_scale.c
ccAsNCAQ.s
libavfilter/vf_scale.o
libavfilter/asink_anullsink.c
ccBKzhtx.s
libavfilter/asink_anullsink.o
libavfilter/vf_overlay.c
ccAOKvOQ.s
libavfilter/vf_overlay.o
libavfilter/vsink_nullsink.c
cccsEX5U.s
libavfilter/vsink_nullsink.o
libavfilter/formats.c
ccy8VBMp.s
libavfilter/formats.o
libavfilter/vf_settb.c
ccrTHBEm.s
libavfilter/vf_settb.o
libavfilter/vsrc_color.c
ccXg6Mno.s
libavfilter/vsrc_color.o
libavfilter/vf_drawbox.c
cc8wJSdK.s
libavfilter/vf_drawbox.o
libavfilter/allfilters.c
ccObJKKj.s
libavfilter/allfilters.o
libavfilter/vf_crop.c
ccjntUBq.s
libavfilter/vf_crop.o
libavfilter/drawutils.c
cc37SuI5.s
libavfilter/drawutils.o
libavfilter/asrc_anullsrc.c
ccdfME6L.s
libavfilter/asrc_anullsrc.o
libavfilter/vf_pixdesctest.c
cch5CtFw.s
libavfilter/vf_pixdesctest.o
libavfilter/vsrc_buffer.c
cc8e2NDb.s
libavfilter/vsrc_buffer.o
libavfilter/vf_setpts.c
ccnVs3fe.s
libavfilter/vf_setpts.o
libavfilter/avfiltergraph.c
ccFZVj3t.s
libavfilter/avfiltergraph.o
libavfilter/x86/gradfun.c
ccukJxbk.s
libavfilter/x86/gradfun.o
libavfilter/vsrc_movie.c
ccvrbOnI.s
libavfilter/vsrc_movie.o
libavfilter/vf_slicify.c
ccQ3HzPI.s
libavfilter/vf_slicify.o
libavfilter/vf_aspect.c
ccOfGUB7.s
libavfilter/vf_aspect.o
libavfilter/vf_pad.c
ccjLBY8h.s
libavfilter/vf_pad.o
libavfilter/vf_null.c
ccFnQ88B.s
libavfilter/vf_null.o
libavfilter/vsrc_nullsrc.c
ccESa7p0.s
libavfilter/vsrc_nullsrc.o
ar
libavfilter/libavfilter.a
ranlib
libavfilter/libavfilter.a
install
lib/libavfilter.a
ranlib
lib/libavfilter.a

Is this enough?

Nope, because we only consider *build time* composition.

Two important things to consider:

- run time composition
- license granularity

Run time composition

- dynamic linking (run time)
- dynamically loading files
- RPC/networked services

License granularity

Licenses can also apply to parts of files:

> ** Kuhn's copyrights are licensed GPLv2-or-later. File as a whole remains GPLv2.*

(source: `wget.c` in recent BusyBox)

> ** The md5_crypt() function was taken from freeBSD's libcrypt and contains*
> ** this license:*
> ** "THE BEER-WARE LICENSE" (Revision 42):*

(source: `libcrypt/md5.c` in uClibc)

Conclusions

- licensing in Open Source software projects is not always as clear as we would want to
- static source code analysis is not enough to discover potential issues
- tracing a build will give us a lot better information and help better determine licenses

Questions