Armijn Hemel
Tjaldur Software Governance Solutions

April 5, 2012

About Armijn

- using Open Source software since 1994
- MSc Computer Science from Utrecht University (The Netherlands)
- core team `gpl-violations.org` since 2005
- owner Tjaldur Software Governance Solutions since May 2011

Subjects

- very brief overview of license violations
- problems in binary code clone detection
- open questions in binary code clone detection

License enforcement

- Europe (Germany, France) & USA
- focus is on GPLv2 and LGPLv2/2.1
- done by companies (Nokia, Red Hat) and individual developers and projects (Harald Welte, BusyBox, XviD, etc.)

It is about copyright, not about patents!

gpl-violations.org

Founded in 2004 by Harald Welte (copyright holder in the Linux kernel) to take on GPL license violations by:

- education
- documentation
- legal action

I have been active with `gpl-violations.org` since 2005.

So far we've had several hundred cases (most of them settled) and fixed many more using informal pressure.

How gpl-violations.org works

1. we get a report via private email, public mailing list, chat, rumours, SMS, or our own research
2. if there is reasonable doubt about compliance of a device we do a test purchase to confirm the violation
3. if we confirm a violation we send a “cease and desist”

There are many false reports: a lot of people don’t understand the license(s).

Our main focus is on consumer electronics (one of the biggest markets out there).

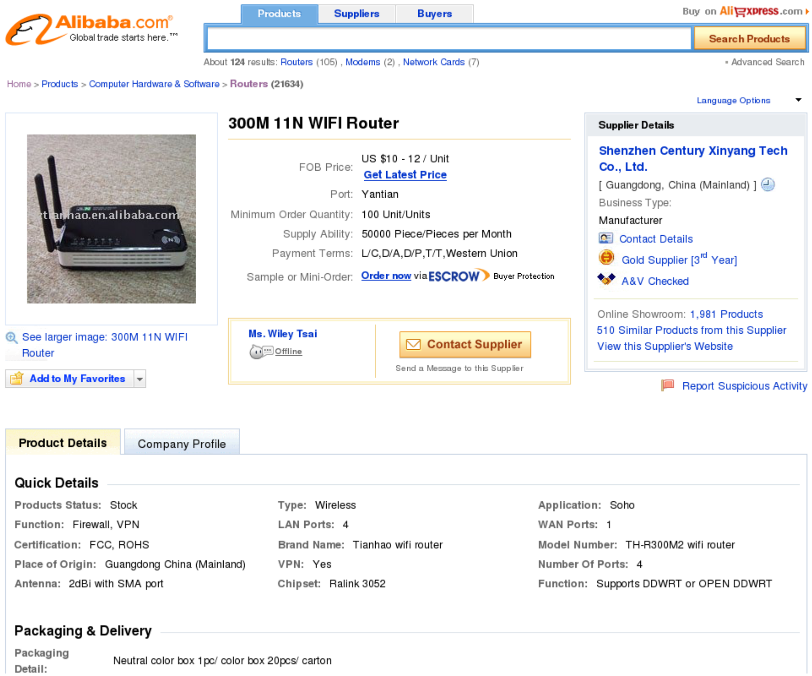
Consumer electronics: the truth

Almost everything is purchased. Making everything yourself is commercial suicide:

- extremely thin margins
- cut throat competition
- quality is less important than price
- (ultra) short term thinking: companies don't know if they will still be in business in 6 months from now
- "cowboys"

It's like Nike: don't do any production, just marketing and sales.

In my experience typically more than 95% (or more) is reuse of open source software (with/without modifications)



About **124** results: Routers (105) , Modems (2) , Network Cards (7)

Advanced Search

Home > Products > Computer Hardware & Software > Routers **(21634)**

Language Options

See larger image: 300M 11N WIFI Router

Add to My Favorites

300M 11N WIFI Router

FOB Price:	US $10 - 12 / Unit **Get Latest Price**
Port:	Yantian
Minimum Order Quantity:	100 Unit/Units
Supply Ability:	50000 Piece/Pieces per Month
Payment Terms:	L/C,D/A,D/P,T/T,Western Union
Sample or Mini-Order:	**Order now** via ESCROW Buyer Protection

Ms. Wiley Tsai
Offline

Contact Supplier
Send a Message to this Supplier

Supplier Details

Shenzhen Century Xinyang Tech Co., Ltd.
[Guangdong, China (Mainland)]
Business Type:
Manufacturer
Contact Details
Gold Supplier [3rd Year]
A&V Checked

Online Showroom: 1,981 Products
510 Similar Products from this Supplier
View this Supplier's Website

Report Suspicious Activity

Product Details | Company Profile

Quick Details

Products Status: Stock	**Type:** Wireless	**Application:** Soho
Function: Firewall, VPN	**LAN Ports:** 4	**WAN Ports:** 1
Certification: FCC, ROHS	**Brand Name:** Tianhao wifi router	**Model Number:** TH-R300M2 wifi router
Place of Origin: Guangdong China (Mainland)	**VPN:** Yes	**Number Of Ports:** 4
Antenna: 2dBi with SMA port	**Chipset:** Ralink 3052	**Function:** Supports DDWRT or OPEN DDWRT

Packaging & Delivery

Packaging Detail: Neutral color box 1pc/ color box 20pcs/ carton

Problem source: supply chain

License violations are often a direct result of a mistake made in the supply chain:

- chipset vendors
- board makers
- SDK ("Software Development Kit") vendor
- reference design makers
- product customizers
- "labellers"

The "labellers" get sued and are responsible, even though they add/modify the least amount of code!

Industry responses to enforcement

- extreme levels of frustration (problem doesn't go away by throwing money at it)
- they don't care about licenses, they just want to sell a product. Licenses are a nuisance that needs to be dealt with.
- a single enforcement case will make no change to the market (it is too big: a single company getting in trouble is not significant to push for change)
- no ill will. Companies want to fix it and there is a need for tools (cheap, or free) to do "due dilligence"

Tools

Apart from the obvious "industry standard" tools that solve some problems Tjaldur Software Governance Solutions has worked on tools to help solving specific problems in this field.

Goal: let companies do checks themselves, increasing quality and lowering costs.

- Binary Analysis Tool (Apache 2 license, freemium model)
- license scanning tools (leveraging existing tools like Ninka and FOSSology)
- long term: build system integration (preliminary work has been done)

Binary Analysis Tool

- generic extensible pluggable framework for analysing binaries
- binary code clone detection using string comparisons: first extract string constants from the binary, compare it with a large database of data extracted from source code, finally assign a score to packages based on matches

Demo later.

Binary code clone detection

"Finding Software License Violations Through Binary Code Clone Detection" (Mining Software Repositories 2011) - some results have been integrated into Binary Analysis Tool

Very simple, but very effective, method using comparison of string constants:

1. extract strings from source code using `xgettext`
2. store strings in database, using some additional information (file name, package name, version, programming language)
3. extract strings from a binary (different methods per binary to decrease false positives)
4. use statistics to compute a score

Computing a score for a package

1. if the string is unique add the length of the string to the score
2. if it is not unique, decrease the score exponentially depending on the amount of packages it is in
3. determine in which package the string is. In case of internal cloning assign a string (and its score) to a package using a “battle royale”

Weeding out false positives

Extracting strings from binaries in a smart way:

- only use `data` and `rodata` sections from ELF binary, if available
- extract string constants from Java binaries using `jcf-dump --print-constants`
- extract string constants from Dalvik (Android) binaries using Dedexer

This reduces the amount of false positives and increases fidelity.

Not done (yet):

- bFLT (ucLinux, not used much anymore, largely irrelevant)
- .NET

Causes of false positives and false negatives

- `xgettext` is not always correct (can't deal with special characters like NUL)
- some strings are defined using escaping in one package and cloned without escaping in another
- generated source code
- I make a strong distinction between language families because code reuse between packages in two language families is unlikely, but this makes it hard when code in another language is embedded (especially an issue for .NET)
- hard to decide where to distinguish between systems where it is unlikely that code is being reused (Linux kernel and desktop will have extremely little overlap)

Seeing BAT in action

- running a full analysis on a simple firmware
- new viewer (not released yet, still quite bad performance in some cases), to view results of the scan

Using information from ELF dynamic symbol table

Dynamically linked ELF binaries have a lot of information recorded in the binary, including function names:

- `readelf -W --dyn-syms`
- filter out everything that is not a function
- filter out everything that is not local, but is linked at runtime (indicated using `UND`)

Using a similar method as string ranking you can do function name ranking:

- use `ctags` to extract function names from source code
- record function names in database, with meta information
- match function names from binary with strings from database

Unfortunately: no demo, since this is still very early days (few days old) and not very efficient (yet). Test runs with some experimental data are very promising though.

Defeating obfuscation

Few companies hide violations on purpose. For each company there comes a point when they want to give up on obfuscation.

- verbatim string matches are easy to work around, but there could still be enough partial evidence (substring matches)
- changing names of function names has additional risks: you are changing the API of programs/libraries

The biggest cost: testing and making sure that things continue to work as planned.

Next steps: generate signatures from compiled code.

Also, avoiding detection means taking many more steps:

- scrubbing (network)
- hiding services
- locking down machines (network, services, hardware via serial port/JTAG)

Open questions/problems about binary code clone detection

- detecting obfuscated code in binaries (when basic string comparisons simply aren't enough)
- detecting language embedding (interpreters, DSLs) in binaries (if they have been compiled)
- correlating binary code and source code (solved for source to binary using tracing, not from binary to source side)
- complete provenance of binary and source code files, down to the level of single commits (example: individual Git commits) because snapshots from DVCS (like Git) are rapidly replacing normal releases
- reducing false positives in detection: false claims can lead to counter lawsuits, with significant risks

Analyzing build processes

"What Goes into an Executable? Identifying a Binary's Sources by Tracing Build Processes" (sent to WCRE 2011 and ICSE 2012, unfortunatey rejected)

Simple method:

- forget about static analysis
- use *tracing* to instrument the build process
- postprocess output from trace and record dependencies between artefacts and inputs
- get information about relevant inputs (license, copyrights, security information, . . .)

I'm preparing more tooling that implements this (rough prototypes exist).

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` whi ch 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.

Tracing software use case: FFmpeg

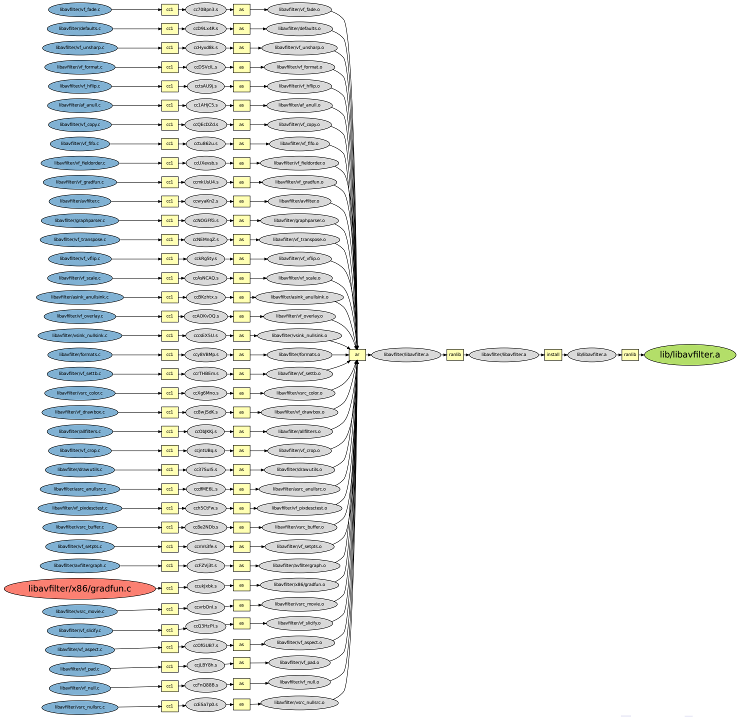
`FFmpeg` is a mix of GPLv2+ and LGPLv2.1+ licensed code. The `configure` script has an option to only use the LGPLv2.1+ sources for a build.

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

Runtime analysis

Most interesting is analysis at runtime:

- libraries at buildtime might be different to runtime (dynamically linked executables)
- there might be dependencies that are undetectable using static analysis or at buildtime (`dynload()`, web services, . . .)

Q & A