Compliance engineering in the embedded industry

Armijn Hemel

April 24, 2009

About me

Professional:

- 1996-2006: computer science at Utrecht University
- 2004-2006: MSc thesis: NixOS
- 2005-present: `gpl-violations.org`
- 2006-present: board member of NLUUG (`http://www.nluug.nl/`)
- 2006-present: Chief Random Projects at Loohuis Consulting

A word from our sponsors: Loohuis Consulting

- specialized hosting
- web development (AJAX and other buzzwords)
- GPL license compliance engineering
- UPnP security
- router/embedded security advice

More info: `http://www.loohuis-consulting.nl/`

`gpl-violations.org` background

`gpl-violations.org` was founded in January 2004 by Harald Welte (Netfilter) out of frustration:

- lots of devices using his software were not GPL compliant
- snubbed/ignored by companies when he requested source code
- soft approach cost tons of time, with few results

A lot of people would give up, fearing the unexplored legal route. Harald saw it as an interesting challenge.

A lot of cases have been handled in the last few years.

How gpl-violations.org works

First a report comes in:

- legal@lists.gpl-violations.org mailinglist (public)
- license-violation@gpl-violations.org (private)
- forwards from FTF (private)
- community projects (OpenWrt, ...)
- own research and watching known/past offenders

After verification (through a test purchase and/or compliance engineering) Harald's lawyers send a cease and desist letter to companies, or we hand it off to the proper copyright holder(s).

Hanlon's razor

> *"Never attribute to malice that which can be adequately explained by stupidity."*

Most companies we deal with don't violate licenses on purpose:

- are not the real manufacturers
- are no engineers (and can't check manufacturer claims)
- no clue: "there is no software in this device"

Between a rock. . .

After doing this for a few years we have found that companies:

- often don't respond (at least initially)
- don't verify claims (almost every time)
- snub us (very often)
- try to drag on cases indefinitely (very often)
- don't learn from past mistakes (nearly always)

But:

- companies should know what they sell

. . . and a hard place

There is also the "community", that:

- expects us to perform miracles and have source code available within hours (too often)
- say we are money grabbing bastards (sometimes)
- think we should cut companies some slack, because they increase the market share of Linux (every now and then)

But:

- Harald's code is not owned by the "community"
- market share without freedom is not why the code was written

Violations aftercare

`gpl-violations.org` is not about extorting money from companies. It is about creating awareness.

To increase awareness we:

- refer to FTF
- give pointers to additional documentation (risk grid, etc.)
- refer to companies who can help to change processes

Unfortunately, not many companies actually follow up on our advice. So we keep bugging them until they listen.

Compliance engineering

Compliance engineering falls down in a few categories:

- physical license compliance
- device analysis
- firmware analysis

The first is handled by the lawyers, the latter two are handled by me.

Physical license compliance

Various licenses (GPL, LGPL) require a copy of the license shipped with the software:

- physical copy (manual, leaflet)
- digital copy (storage, in menus in the user interface)

Many people regard checking this as anal, but it is a great litmus test for further compliance testing!

Device analysis

Analysing a physical device is done primarily via two methods:

- analysis via the network
- physical modifications

The first method requires nothing but a network connection and the proper software tools, the second method might require a soldering iron and some hardware parts.

Analysis via the network (1)

A portscan with fingerprinting can reveal the OS.

```
# nmap -O -P0 10.0.1.15
Starting Nmap 4.53 at 2009-04-06 18:46 CEST
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 00:00:00:00:00:00
Running: Linux 2.6.X
OS details: Linux 2.6.11 - 2.6.22
```

Pro:

- a positive match for Linux is a strong indication the device is running Linux
- it reveals which services are running

Con:

- techniques such as "scrubbing" can hide information

Analysis via the network (2)

Analysis of services can reveal a lot:

- telnet (can grant direct access to the running system)
- services (webserver/UPnP daemon/others) can reveal OS in server banners or give other circumstantial evidence

Web interface on devices can reveal a lot too:

- logs containing version numbers and command output

Physical modifications

Many devices provide some external access to the device:

- serial port
- JTAG

Serial ports are often left overs from debugging or serve as a management console. Connecting will often give direct access to the device.

JTAG is for the real experts who know what they are doing.

Before serial port and JTAG can work often some soldering work and investigation ("what does this do?") needs to be done.

Boot loaders

A boot loader is a small program that loads and start the kernel. Most, but not all, are proprietary. Popular GPL licensed boot loaders are:

- ARMboot
- u-boot
- RedBoot
- PPCboot
- GRUB

Often these can only be detected by analysing the kernel(sources) and using the serial port.

Getting boot loaders GPL compliant is a lot of trouble.

Firmware analysis

A firmware appears as a blob of data, but inside you can often find:

- compressed data (kernel image, file systems, binaries)
- kernel image
- file systems

in various combinations (depending on the device).

Partition boundaries

The boot loader in a device often expects to find the kernel, file systems and other data at a fixed offset.

Padding is used to make sure a partition starts at the right place. This is usually indentified by thousands of identical characters."

Quick method I use a lot:

- look for padding (`0x00` or `0xff`)
- look for known file systems or compression identifiers just after the padding ends

Compression

A lot of data is in compressed form in the firmware. Common compression methods are:

- ZIP
- gzip
- bz2
- 7z
- LZMA

Compressed files in a firmware can be recognized by a certain start pattern ("magic").

File systems

File systems in common use are:

- SquashFS (with/without LZMA compression, in various forms)
- ext2/ext3
- Minix file system
- FAT/vfat
- romfs
- cramfs
- jffs2

Analysing binaries

- program names/versions
- extract visible strings (debug strings, output strings) with `strings`. The order in which they appear in sources and binaries is often the same.
- find dynamically linked libraries (dependencies) with `readelf`. Some libraries are GPL-licensed.
- extract symbol names with `nm`. This only works with binaries that are not stripped.

Some binaries (on uClinux) have compression (bFLT with gzip). Some extra work needs to be done to unpack these.

Is this decompilation? I don't think so!

Demo time

You will see:

- discovery of various parts (`vi`, `hexdump`)
- extraction of kernel
- extraction of a file system
- analysis

Difficulties

- encryption (TiVoization)
- non-standard compression (SquashFS + LZMA with slight modifications to either)
- most devices are different (different offsets, different compression), even if they use identical hardware

More technical information

More technical information can be found in the GPL compliance engineering guide:

`http://www.loohuis-consulting.nl/downloads/compliance-manual.pdf`

Bug reports and more information are always welcome.

Future work

None of the analysis is currently automated: it is faster for me to do by hand.

Automation would be nice (FOSSology?), but:

- lack of time
- lack of motivation (after all, this is what we get paid to do)

Getting automated:

- Someone would have to pay for it

Contact information

Loco (Loohuis Consulting)
Jasmijnstraat 3
3551 SP Utrecht
The Netherlands
Phone: +31 (0)30 291 9805
E-mail: armijn@loohuis-consulting.nl
Web: `http://www.loohuis-consulting.nl/GPL/`