Automotive Software Governance and Copyleft

Mark Shuttleworth* Eben Moglen†

October 12, 2018

Introduction

Our purpose in writing this paper is to show how new capabilities in the free and open source software stack enable highly regulated and sensitive industrial concerns to take advantage of the full spectrum of modern copyleft software, including code under the GNU General Public License, version 3 ("GPLv3"), and to manage their obligations under those licenses in ways that are commercially sound.

Software embedded in physical devices now determines how almost everything—from coffee pots and rice cookers to oil tankers and passenger airplanes—works. Safety and security, efficiency and repairability, fitness for purpose and adaptability to new conditions of all the physical products that we make and use now depends on our methods for developing, debugging, maintaining, securing and servicing the software embedded in them. These methods of "software governance," are to 21st-century technology what materials science and quality assurance practices were to 20th-century industrial activity. They are now a crucial "hidden input" to industry's ability to make, and government's ability to regulate, everything we use.

Few products encapsulate both the challenges and the possibilities in this area like the automobile. We stand on the verge of a transformative set of changes brought about by autopilot software capable of autonomously controlling some forms of "driving." But even now, the automobile is already a constellation of computers and an ecology of software. Even as currently designed for human operation and control, a contemporary car can include dozens of computers receiving input from hundreds of sensor devices and actuating everything from brakes to entertainment systems, all under the control of thousands of software components. Cars have long lifetimes, necessitating both scheduled and episodic maintenance. Safety testing and regulation of software involved in vehicle operation is extremely difficult, even given perfect trust between manufacturers and governments, which recent events have shown cannot be preserved. In addition, manufacturers have legitimate trade-secret and regulatory compliance interests which may conflict with the economic and social interests that

*Mark Shuttleworth is CEO of Canonical Ltd., which distributes the all-FOSS Ubuntu Core system.

†Eben Moglen is Professor of Law at Columbia Law School and Founding Director of the Software Freedom Law Center.

are served when the software in vehicles can be inspected, tested, and repaired by the largest number of qualified parties. Our 20th-century experience with automobiles showed the value for manufacturers in following the innovations created by car owners themselves, because people could learn to fix, adapt, and reuse automotive technologies by working "after-market" on cars.

For these reasons, society and the industry have much to gain from the use of a software governance system based on "free and open source" software ("FOSS") in automobiles, such as Ubuntu Core—produced by Canonical Ltd.—which creates a framework that can be used to balance potentially competing objectives. FOSS is software produced and distributed under rules that give purchasers and users both the legal rights and technical enablements (like possession of source code and a means to install and use fixed or modified versions of programs) necessary to study, improve and share. FOSS has become the most important infrastructure material in software over the last generation. It includes the Android operating system in the vast majority of the world's smartphones and tablets, and the GNU/Linux operating system dominant in server computers. It enables the utility computing power around the world we call "the cloud." It comprises the software infrastructure on which the world's digital "platforms" like Google, Facebook and Twitter run. FOSS is everywhere.

The pace of innovation in the automotive sector, as in other technology-centred sectors, has accelerated to the point where it is only economically feasible to compete by using shared and open code. FOSS has become central to the race to create better automotive experiences, and every established manufacturer and startup in the sector is dependent on a body of shared code under FOSS licenses. However, the automotive industry has tended to limit its own access to widespread innovation by avoiding FOSS under copyleft licenses which facilitate user modification, for lack of an effective means to manage those obligations in a commercially satisfactory manner.

In the automotive software environment, new technical capabilities in FOSS can help to solve the profound engineering and social problems of governance, security and liability from which we already suffer and which the rapid technological changes occurring in the industry will aggravate. These newly available capabilities and methods of software distribution can:

- Guarantee manufacturers', owners' and regulators' precise knowledge of which version of what software is installed in any vehicle at any moment;
- Enable efficient, secure and reliable updates to be performed "over the air" or offline;
- Limit the access of individual software components with very precise granularity;
- Determine who has serviced or modified that software;
- Restore erroneously or incompletely modified software to a known, reliable state; and
- Even install or revert software upgrades and fixes during vehicle operation.

In addition, these new FOSS capabilities can provide clear and efficient mechanisms to mediate the rights, liabilities and regulatory responsibilities of manufacturers, users and developers addressed in the GPLv3, and thereby allow manufacturers and society to access the full range of FOSS innovation and capture the immense value of user participation in the after-market. This important goal can now be achieved while preserving manufacturers' ability to prevent safety or regulatory violations and limiting their liability for accidents or unsafe tampering.

This essay shows how a specific, existing form of FOSS software distribution, Ubuntu Core and "snap" technology, can achieve these goals under present technological conditions. We believe that the methods of packaging and distributing software we describe here prove a software governance concept sufficient to solve many of the basic problems in the automotive software environment. These approaches safely preserve the technical and social value of the "right to repair" and the "right to tinker," from which society as a whole stands largely to gain.

Automotive Software Environments

Contemporary automobiles are networks of heterogeneous computers, often numbering in the dozens, tied to hundreds of input sensing devices and output displays and actuators controlling combustion, steering, braking, and all the other physical behavior of the car. Some of these computers are embedded in components sold to the OEM by one of its first- or second-tier suppliers, who also provide all the software those computers run; some are designed and placed in the vehicle by the OEM itself, running application software it has developed internally or purchased from an upstream software supplier. Processors in both of these classes can be special-purpose machines, designed to run a narrow suite of software, as well as general-purpose processors like those in laptops or tablets, running a conventional operating system and ancillary program code underneath application-layer code providing the major functionality required for, e.g., entertainment, console instrumentation display, or climate control.

In-Vehicle Networks

These heterogeneous computers, running diverse repertoires of software, are interconnected by equally heterogeneous networking structures, ranging from analog switching over dedicated wires to internal TCP/IP-based networks, to wireless connections to the public mobile Internet. There is no general systems engineering discipline that governs this in-vehicle network. Subsystems as different as door locks and ignition systems may receive control signals from the same computer receiving wireless inputs from a key fob, for example. Computers controlling passenger-compartment devices serve multiple purposes. They run software that users can productively modify (for media playing over entertainment systems, or receipt and placing of cellular phone calls, for example), and also software controlling actuation of steering or braking, which would raise serious public safety and liability concerns if modified.

Connected Cars

The vehicle's connection to external networks potentially carries telemetry information crucial to collision-avoidance and traffic management, but also capable of fundamentally compromising passengers' privacy. Software modifications designed to protect passenger privacy should be enabled, while safety-critical uses of similar data must not be blocked or inhibited. In the near future, as "smart road" technology increasingly appears on some, but not all, streets and highways, the software environment of the automobile will be decisively affected, moment to moment, by the particular route on which it is traveling.

The functional and organizational complexity of this array of computers and network segments demands powerful practices and mechanisms of software governance. Product engineers, repair workers, vehicle owners, regulators, and liability lawyers all have different but important interests in knowing what software is installed on which computers in each car, how it came there, who has modified or upgraded it, and so on. The alternative is chaos, which is largely the present condition of the industry.

SNAPs and Their Governance Properties

The general-purpose computers that are part of the in-vehicle network each run a standard computer operating system and various application programs atop that OS. There are several ongoing initiatives to encourage the use of FOSS software stacks across all these general-purpose computers. One of us (Shuttleworth) is the CEO of Canonical, which produces Ubuntu, a FOSS operating system, that is widely used in autonomous vehicles and industrial systems.

New capabilities in the Linux kernel have catalyzed a wave of innovation in mechanisms to package and deliver applications, using containers and cryptography to address long-standing operational issues. Some of these packaging and governance technologies are particularly valuable in responding to the issues in automotive systems. Here we focus on the solutions made possible by the style of software distribution called "snaps".

Snap format files encapsulate an application program and all its dependencies—both the libraries it links to and the static data and configuration files it requires to execute—in a compressed, read-only filesystem. Once installed, the code and data representing the application cannot be changed. An entire system, consisting of a "kernel snap" for the OS kernel, a "core snap" for basic system facilities, and a series of application program snaps can thus be "snapped together" to create an entire system in a verifiable base state, in which all the system's major components and applications are isolated and guaranteed to remain uncorrupted and uninvaded by "malware" throughout their installed life.

Because each snap incorporates all of an individual application's code and data dependencies, an upgrade or rollback of that application can occur with assurance that this change will not break any other installed snap or the system as a whole. Returning to a past state of any or all of the applications in the system does not pose a risk of an incompatible or incoherent installation.

The installation and management of snap files in the software stack is handled by a software governance agent, called "snapd." The snapd process runs in the background, acting to install, rollback, and remove snap files containing versions of the OS kernel, system libraries, and application programs. The governance agent uses digitally-signed documents, known as "assertions," to determine which snap files to load. In a typical automotive context, the snapd in each computer on the vehicle network would only install snaps whose defining assertions are signed by the OS vendor, the relevant component manufacturer, or the vehicle OEM itself. Each snapd in the vehicle network can report in real time, on request, the complete software installation state, and the installation and modification history, of every application package on the system. Assembling a complete software inventory and maintenance history across the entire vehicle network is equally simple. Software governance at the package level using snaps does not imply that all the software governed is FOSS. Snapd and other programs that enable the forging and distribution of snap files are licensed under FOSS terms, but the software distributed in the snap package format may be licensed under any terms.

Snap packaging also enables strict governance of communication between software components installed on the same device. Each program or collection of programs contained in a snap receives rights to access other programs, devices or computers on the network through "interfaces" managed by snapd. The list of interfaces that the application in the snap can "connect" in operation is determined by another digitally-signed assertion that snapd reads and verifies whenever the application in the snap is executed.

This means that the impact of modification can be limited to a particular snap, and the set of components with which a modified snap can communicate may be strictly governed by the device manufacturer to manage the balance of rights and liabilities.

Let us assume, for purposes of illustration, a single computer attached to the in-vehicle network of an automobile. This computer runs the information display and sound system in the passenger compartment. It has installed a media controller, such as the well-known package Kodi, for playing video and audio files that are passenger entertainment, acquired from physical storage devices in the automobile, or over the mobile Internet. Also running on the same system are the OEM's applications that offer vehicle control and operation displays—reading data from other computers linked to engine-control, steering and braking systems; navigation and traffic data from the "smart road" components around it; data displayed from cameras embedded in the vehicle body, etc. The OEM's display applications will have "interfaces" allowing communication with other computers in the vehicle, including in situations where the car's operation can be affected, as when the driver activates a positive control on a touchscreen that the media player might also use to display an entertainment video. But the interfaces for the Kodi instance will only allow the media player to access some of the car's video displays and audio system. Snapd ensures that Linux will enforce these strict governance rules. It will also permit the media player to receive media files over the car's mobile internet connection, without permitting access to other segments of the in-vehicle network, preventing bugs or malware in the media player software from affecting vehicle operations.

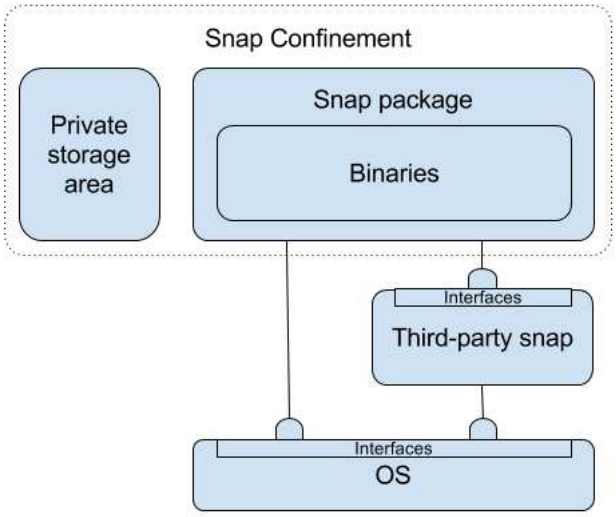

Figure 1: Interface Governance in Snapd

Interfaces governed by snapd—whose definitions are contained in assertion files authenticated by the digital signatures of OEMs, component manufacturers or software vendors—provide very fine-grained control over access rights in the in-vehicle network, allowing carefully-tailored protections against both inadvertent and deliberate software modifications that could affect safety, privacy, or manufacturers' liability for accidents.

Using snap packaging for the software in general purpose in-vehicle computers is a simple step that by itself achieves fundamental goals in improving software governance:

- *Accountability:* All computers can report the authenticated state of all installed software. Any overall state of the software on all systems in the network can be saved for later restoration;
- *Maintainability:* Any individual software component, from the system kernel to individual applications, can be upgraded to a new install or reverted to a saved prior state, with assurance that no other software component will be broken by the change;
- *Security:* Each software component is authenticated by digital signature identifying the source of the package (upstream OS vendor, tier 1 supplier, OEM, OEM's product app store, etc.) Fine-grained access control rights to devices, in-vehicle network data sources, control subsystems, external networks, etc. are defined for each package, by digitally-signed interface assertions continuously enforced by the governance engine on the running system;

- *Adaptability:* Multiple states of each software package throughout the vehicle, and overall states representing a snapshot of all the software on all systems, can be saved and restored with assurance that each version of each package will operate correctly. All such state changes are auditable for maintenance and other purposes. The vehicle's software environment therefore gains comprehensive adaptability: it can be changed both globally and locally to respond to operating conditions, to deal with hardware failures, and to capture user innovation.

Copyleft and the Right to Tinker

The adaptability in software governance brought about by this simple change in software packaging bears directly on the ability to capture user innovation in vehicle software, because it also allows us to govern user-modified and other experimental versions of FOSS software effectively.

Capturing user innovation can only occur if we enable car owners to modify the vehicle's software environment. This openness to downstream innovation is the heart of the success of FOSS as a model for making software. The history of automotive technology also shows the enormous importance of user ingenuity in seeding and stimulating technological advancement.

Preserving the "right to tinker" yields larger social gains, beyond the economic value to manufacturers of user innovation. The free after-market in repair services, which includes software maintenance along with other forms of vehicle service, is a source of employment and small business activity. As ride-sharing and fleet services, such as Lyft, Uber and Zipcar, seek to limit individual ownership and operation of self-driving cars, in part on the ground that other entities lack the expertise to perform such complex software maintenance,[1] it becomes apparent that the subject also involves basic competition law issues.

The copyright licenses that make FOSS possible can be divided into two categories. So-called "permissive" licenses allow programmers to place under any license of their choice programs made in part from FOSS parts. This licensing structure maximizes the choices available to software developers. The category of "copyleft" licenses, on the other hand, requires programmers and firms using copylefted FOSS parts to release their own work under the same or equivalent copyleft licenses. This licensing structure emphasizes the rights of *users* to receive program source code and the other materials necessary to enable experimentation, improvement, and adaptation to new purposes. The most widely-used and influential copyleft licenses are in the family of the GNU General Public License, published by the Free Software Foundation and applied not only to its GNU operating system, but to hundreds of thousands of other computer programs used in all technical and social contexts.

GPLv3, a license promulgated in 2007 after lengthy public review and discussion, requires not only that GPLv3 programs and all works based on them enable users to modify and share their modified versions, but also requires that when such programs are distributed embedded in a "user product" (which includes an automobile) that

[1] See, for example, www.sharedmobilityprinciples.org, point 10.

"installation information" be available to enable the user to install and operate modified versions of the program in the product itself. (One of us, Moglen, was counsel to the Free Software Foundation during the drafting of GPLv3, and directed the subsequent public discussion process.) This protection of the "right to tinker," built into the DNA of copyleft FOSS, has two closely-linked objectives:

1. To protect users' rights to understand and control the digital technologies that increasingly undergird their lives; and

2. To enable businesses to derive concrete economic value from user innovation, to provide a "virtuous circle" in which the profit motive in industry supports the technological and civil freedoms of individuals.

But in the automotive software context, the requirement to permit user installation of modified versions of software has been a serious obstacle to adoption of GPLv3-licensed FOSS. Vehicle manufacturers are understandably concerned with the liability consequences if user-modified software results in malfunction or accident. Maintainability can be adversely affected by unexpected interactions between user-modified software and other components on the network.

For these and other reasons, architects of FOSS software environments for automotive use have largely eschewed GPLv3-licensed programs.

What is now becoming clear is that a very large portion of modern FOSS will be inaccessible to those who are unable to include GPLv3-licensed programs in their software solutions. Many FOSS components are moving to GPLv3 or depending on GPLv3 software. The cost of maintaining non GPLv3 forks of large portions of the FOSS stack is prohibitive. In parts of the automotive software landscape, the ability to use the full range of FOSS including GPLv3 code is important to stay competitive—for faster and more productive development, access to new capabilities, and access to a wider ecosystem. For these reasons, it is attractive to manufacturers to explore ways in which they can integrate GPLv3-licensed software into areas of their practice, managing the resulting responsibilities in a commercially reasonable fashion.

Wholesale avoidance of legal arrangements intended to protect users' rights is an ominous solution to the problems presented. In order to help manufacturers' capture the value of user innovation without accumulating unnecessary risk, and to protect users' rights in automotive technology—which has been a major contributor to human freedom for a century—a software governance approach that doesn't involve foregoing all copyleft "right to tinker" software is desirable.

The combination of accountable and adaptable governance provided by snap packaging makes workable solutions possible, inexpensively.

Suppose a car owner, who is also a software developer and a creative tinkerer, wishes to install a modified media center software package, to allow her and her family to record choral music together on the way to school, or to achieve some other worthy, idiosyncratic goal. She requests from the OEM customer engagement website, using her car's Vehicle Identification Number, "installation information" for the passenger compartment media player software, along with the source code

for the OEM's shipped version of the program she wishes to modify. In addition to the source code and instructions to install the modified version she makes, she receives a signed pair of assertions from the OEM, one allowing her to install her modified version of the media management software in the relevant computer in the in-vehicle network, and one stating the interfaces, that is, the access rights of the modified program. These signed assertions are necessary for the snapd in the target computer to install and execute her version.

Once installed, the car owner's media player will do whatever she has modified it to do. Where the OEM judges it necessary for the safe operation of the vehicle and its network, her version may, however, lose some access rights on the in-vehicle network that the manufacturer's software possessed. The fine grain of interface access rights provided by the snapd governance agent can thus provide further isolation and security when it is running user-modified code, guaranteed under the snap packaging paradigm to cause no other program code to be modified, to break, or to perform differently because of the presence of the user-modified program. Such a structure of modification permission can be operated by the OEM consistent with the requirements of GPLv3. The OEM can publish an authenticated record of the installation permission issued, indexed by the Vehicle Identification Number—without publishing the car owner's personal information—so that public and private parties can be assured that no surreptitious modification of vehicle software occurs.

More can be done, at no additional cost of effort or expense, through snap-based software governance. When the vehicle is serviced, or a warranty claim is made, the user's modifications can be automatically rolled back, so that only the OEM's base state and maintenance upgrades to vehicle software are active. In the event of questions about liability for accident, the OEM can prove both the state of the software in the car at the time of accident, and the extent to which operational changes were the result of user modification, limiting its own responsibility. The ability to reach a known state by reliably reversing temporarily the user's modified versions of programs, wherever they may be installed in the vehicle, can also be applied in other, more selective ways: by returning to known state when operating in hazardous conditions, or under autonomous control. More sensitive geographic alterations in software state are also possible, even including—for example—a wider latitude for the car owner to modify software when operating on her own property than when the vehicle is operated on public roads. Any such arrangement "just works" because of the governance properties of snap packaging.

Conclusion

The issues of software governance in automobiles represent the leading edge of similar issues throughout society, as the automotive industry once again explores the frontiers of technology and powers social change. Simple changes in how software for cars is packaged and distributed, such as snaps with digitally signed assertions and secure, mediated interfaces, can make an enormous difference in increasing reliability, security and maintainability of vehicles, and in providing for valuable forms of user innovation, through tinkering, adaptation and improvement.