



Dispelling Myths about the GPL and Free Software

John Viega and Bob Fleck
Cyberspace Policy Institute

“I cannot help fearing that men may reach a point where they look on every new theory as a danger, every innovation as a toilsome trouble, every social advance as a first step toward revolution, and that they may absolutely refuse to move at all for fear of being carried off their feet.”

— Alexis de Tocqueville

On June 10, 2002, the Alexis de Tocqueville Institution released a white paper entitled “Opening the Open Source Debate”. That paper served to perpetuate some common myths about the open source and free software communities in general, and the General Public License (GPL) in particular. It also introduced some new misconceptions we had not previously seen in arguments.

One of the most disturbing aspects of the hype surrounding the de Tocqueville report is the shockingly wrong implication that free software may leave the United States government open to more risk from terrorist organizations than if using proprietary software. In this paper, we will dispel this myth, along with thirteen other misconceptions that are often used to scare corporations and governments away from using free software, particularly that licensed under the GPL.

We do not purport to take sides in this battle. The authors have developed programs distributed under a variety of licenses, both free and commercial. Our goal is to provide objective information to dispel common misconceptions, hopefully enabling corporations, governments and individuals to make better decisions about what software they should deploy in any particular situation.

Myth #1: Arguments against the GPL apply to all “Open Source” software.

In presenting business arguments against software licensed under the GPL, some critics seem to be arguing against all software that can be obtained and used for no charge in source code format. Generally, critics will use the terms “GPL source code” and “open source software” interchangeably. There are those who feel such confusion is intentional— that proprietary software vendors are trying to push consumers to use their software over free alternatives, by suggesting that any issues that may hinder the adoption of software licensed under the GPL actually apply to all free software.

A less cynical argument is that such confusion is unintentional, and is due to the fact that the relevant lexicon in the technical community does not map to the intuitive definitions many people ascribe to various terms. For example, the intuitive definition of “free software” for many people is that the end user be allowed to obtain and use the software without charge. That is, when most people contemplate the term “free software”, they think about “free as in beer.” The technical community largely uses this term to mean “free as in speech”. In this community, software is considered to be free if people to whom a program has been distributed can examine, run, distribute and modify a program without restriction. Programs distributed without source code can not be considered “free” under this definition, because source code is widely considered a precondition for the masses to examine or modify a program.¹ While the GPL is the premiere “free software” license, there are many other licenses that meet this definition. The Free Software Foundation (FSF), which is responsible for the GPL, maintains a list of free licenses on its web page (<http://www.gnu.org/licenses/license-list.html>).

Another area of confusion is the term “open source”. The intuitive definition is that source code is available, and nothing more. After all, if an object is “open”, then, while we can see into the object, there is no guarantee that we can do anything with what we see inside. For example, a house with an open door does not give us the right to enter the house or take the door. In the technical community, we have “open standards” which are developed in the public eye, but may still require implementers to pay to get a copy of the standard.

The Open Source Institute (OSI) has popularized a more restrictive definition of this term. Their intent was to provide a new term to replace the confusing term “free software”, so that the term could unambiguously refer to its more natural meaning. This goal failed, because the term “free software” is still commonly used in the sense of free beer. Additionally, the term created a new ambiguity, as “open source” would have been the most natural way to say, “the source code is available” without implying anything else about the software. Worse yet, the FSF-approved definition of the term “free software” and the OSI definition of the term “open source” are not quite the same; the terms cannot be used interchangeably.

We acknowledge that the lexicon in this area is likely to remain muddled for some time. However, in order to avoid ambiguity, we will use the following terminology in our discussion:

Open box software is any software for which anyone can look at the source code. No implications are made as to how a person who has obtained the source code may use the source code. For example, Microsoft could give away the source code for Microsoft

¹ As we will discuss later, there are people with the skills necessary to examine and modify the program without the source code. However, the free software movement seeks to preserve the freedoms of those who possess the original work, and give those people the ability to pass those freedoms onto anyone else.

Word, yet forbid anyone from changing or compiling the code. In that case, the software would still be “open box”.

Free software shall be any software that can be examined, run, distributed and modified without restriction by anyone who has received the software. In short, we will ascribe fully to the FSF’s definition of this term (see <http://www.gnu.org/philosophy/free-sw.html>). Despite confusion over this term, it is at least a natural use of the word “free”, and we have never seen a suitable suggestion for alternate terminology.

To complete our terminology, we call any software that can be obtained at no cost *freely available software*. This term does not imply source availability, nor does it imply a right for individuals to modify or redistribute the program.

We shall explicitly avoid using the term “open source”, except when paraphrasing the arguments of others, in which case we shall place the term in quotation marks. In such cases, the term was used to mean “free software” by our definition.

Unfortunately, while the Alexis de Tocqueville Institution report often makes a distinction between software licensed under the GPL with all “open source” software, often it uses the term “open source” when clearly only the GPL is at issue. In such cases, we will do our best to clarify the issue.

Myth #2: *Free software is bad for government.*

In this paper, we will debunk specific arguments that might otherwise provide support to the hypothesis that free software is bad for government. In addition to addressing those individual misconceptions, we would like to demonstrate the interest that governments actually are showing in free software. First, several reports favoring free software have recently circulated around the United States military to favorable review, including “A Business Case Study of Open Source Software”, authored by the MITRE Corporation, which is available online at http://www.mitre.org/support/papers/tech_papers_01/kenwood_software/index.shtml.

Additionally, Peruvian Congressman Edgar Villanueva Nuñez made a convincing argument to Microsoft Peru as to why governments should embrace free software, available at http://www.pimientolinux.com/peru2ms/villanueva_to_ms.html. This document helps dispel many of the myths we discuss in this paper.

Back in the US, the Defense Advanced Research Project Agency (DARPA) is funding the development of security solutions based on free software in its CHATS program (see <http://www.darpa.mil/ipto/research/chats/>). The program funds significant work in OpenBSD, FreeBSD, Linux and OpenSSL. It also funds a handful of cutting edge technologies for automatically finding vulnerabilities in software.

The President’s Information Technology Advisory Committee stated in their October 2000 report to the President of the United States, entitled “Developing Open Source Software to Advance High End Computing”, that they had “concluded that the open

source software model merited investigation because it offered a unique approach to producing robust software through a mixture of public, private, and academic partnerships. By involving these multiple communities, open source development efforts offered the additional promise of allowing the government to leverage Federal funds with existing human infrastructure in the academic and business sectors.”

Additionally, the National Security Agency has developed and released “Security Enhanced Linux” (SELinux), a free extension to the popular operating system that adds strong security properties that don’t exist in off-the-shelf versions of Windows.² Currently, there are several projects aimed at turning SELinux into an easy-to-install distribution, at which point, it is likely to be widely adopted within the United States government, as it better meets security requirements, and is pushed by the NSA, which has been critical in driving information security standards used within the government.

Myth #3: *Security problems in free software are there for any bad guy to find easily.*

The Alexis de Tocqueville Institution report goes so far as to say that governments worried about terrorism should not deploy free software on their networks because terrorists might find and exploit holes.

According to the proponents of this myth, proprietary software is better because security holes are incredibly difficult or impossible to find when the source code isn’t available. This is a huge misconception.

In our daily business, we reverse-engineer compiled programs for our clients looking for security flaws that might impact their business, and have no problems finding such flaws. It’s almost as easy for us to find security problems in a binary as it is for us to find security problems in source code. While we do have automated tools that help our hunt in both cases, we have often done such work manually, with similar success. The only difference is how much time and skill is necessary to find the security problems.³

Admittedly, it does take more time and skill to find security problems in binaries. Nevertheless, there are plenty of people with that time and skill. If those people are targeting any particular organization, then the availability of the source code simply doesn’t matter if there’s a security hole in the code will not make a difference.

Furthermore, the source code to proprietary programs does not always remain secret. In October 2000, Microsoft’s network was penetrated by crackers. Steve Ballmer assured

² There are older versions of Windows NT that have been modified for the government to provide the basic security functionality SELinux provides, but that product was not widely used throughout the government, because it came with no networking support.

³ The report also seems to imply that reverse engineering software is an illegal act. In most cases, under US and EU law, it is perfectly legal, regardless of what the license says. For more information see *The Law & Economics of Reverse Engineering* by Pamela Samuelson and Suzanne Scotchmer:

<http://www.sims.berkeley.edu/~pam/papers/l&e%20reveng5.pdf>

the press that the source code to their Windows products had not been changed, but stated, “It is clear that hackers did see some of our source code.”⁴ Crackers, dishonest employees and even the security problems of corporate partners and customers can all lead to the disclosure of the source code of proprietary software.

This myth is discussed in more detail in Chapter 4 of the book *Building Secure Software*.

Myth #4: *If you want secure software, you shouldn't trust free software, because it is written by untrusted people, and has not been audited by a trusted source.*

The implication of this myth is that non-free software is written by trusted people, and that it perhaps is audited. While many development organizations may like to think their staff is fully trustworthy, the reality of the security industry is that software developers are not to be fully trusted. Furthermore, many development organizations have non-existent or inadequate security auditing procedures for software.

In the past few years, several back doors that were unknown to management were found in proprietary electronic commerce applications, including a well-publicized case where a back door was found in Microsoft's FrontPage 98. Similarly, many developers waste company time and resources building unauthorized “easter eggs”, such as the pinball game in Microsoft Word, or the flight simulator in Microsoft Excel.

At least when source code is available, those organizations that are highly security conscious can pay qualified, trusted auditors to do the work. Without the source code, there are far fewer trusted sources for audits due to the extra skill required. Additionally, with free software projects, there is always the hope that independent auditors will review the code without remuneration, for whatever reason (the “many eyeballs” theory). Commonly, such auditors will be looking for the recognition of their peers.

Myth #5: *Software licensed under the GPL explicitly forbids commercial use.*

This argument does not apply to all free software licenses. Many licenses, including the modern BSD license and the X11 license both permit any use of software provided that the copyright notice stay attached to the original software, and provided that the authors of the program not be used to promote any use of the software without written approval. Indeed, there are many products that are completely proprietary, which were built upon free software distributed under both of the above licenses.

Software licensed under the GPL may also be used commercially. For example, there are many operating system distributions using the Linux kernel as their foundation. Companies use such systems in a commercial context all the time. Such machines run firewalls, and even serve as mail servers for paying customers.

Nothing about the GPL prohibits using software to make money. The confusion comes about because the GPL is designed to ensure that the fundamental “freedoms” associated

⁴ The BBC article *Microsoft Software Stolen* is available at http://news.bbc.co.uk/1/hi/english/business/newsid_993000/993933.stm

with free software cannot be taken away from anyone who receives a derived copy of the software. This means that software vendors can charge for software licensed under the GPL if they wish. However, if someone buys a version of the software, they must have the right to inspect, modify and freely redistribute the software. Whether more than one person will pay for software licensed under the GPL begs the issue of whether such software can be at the core of a practical business model. We will return to this question later.

Myth #6: *The GPL is a virus that “taints” any software that you use with it.*

Critics of the GPL overstate the realities of the situation.

The GPL is the only well-known free software license where this concern might appear valid. Other free software licenses tend to allow derived works to be distributed under a different, more restrictive license.

The GPL does not allow the freedoms that it provides to be revoked by any derived work. If one wishes to start with software released under the GPL and distribute a derived version, then the derived version must indeed be free software. In addition, if you wish to add functionality to a program by copying code released under the GPL into another program, then you are creating a derived work that must preserve the freedoms promised by the GPL. These freedoms must be preserved even when simply *linking* to another piece of software.⁵

However, that is not to say that code licensed under the GPL cannot interoperate with proprietary software. The GPL is firmly rooted in copyright law. The GPL only imposes restrictions on works derived from software licensed under the GPL, where the definition of “derived” is easy to interpret, because it is backed by a clearly defined copyright system.

For this reason, it is possible to create a proprietary program that runs on an operating system licensed under the GPL, without any conflict. For example, there are plenty of people who run the proprietary program VMWare on Debian GNU/Linux. Just because VMWare is available for that platform does not mean the code gets “tainted” by the GPL.

Microsoft has vocally criticized this practice. Ultimately, this may be due to them wishing to adapt code that would otherwise be available only under the GPL. If they can convince more free software projects to adopt licenses that allow derived works to restrict freedoms, then they will have more opportunities to adapt free software into their own products.

⁵ *Linking* two pieces of software roughly means to combine those pieces after they are both transformed into a low-level format; this can be done before the program is run (static linking) or while the program is running (dynamic linking). The GPL does have the virus effect for all instances of static linking, and most cases of dynamic linking, though there may be some cases where the end-user initiates dynamic linking where the “virus-like” property of the GPL does not hold.

Interestingly enough, Microsoft does distribute GPL licensed programs. Their Windows Services for Unix product (called Interix) is intended to help integrate Windows and Unix hosts. In support of that integration, they use many GPL programs, which are distributed on the CD with source. While this fact is not emphasized elsewhere in Microsoft literature, or during their denouncements of the GPL, it is mentioned on the product website <http://www.microsoft.com/windows/sfu/productinfo/overview/default.asp>.

Ultimately, it makes little sense to us for anyone to complain about licensing restrictions imposed by the GPL. First, the vast majority of people who consume software are users, not developers. The GPL has little impact on end-users, requiring only that they make the source code available along with the binaries should they pass a copy on to someone else. For the wealth of organizations out there that simply wish to run software released under the GPL, there are no risks whatsoever. Such people are as likely to want to change a line in the Linux kernel as a line in Microsoft Windows XP. That is, they would prefer someone else do it.

Second, proprietary software companies generally place all sorts of restrictions on releasing derived works (usually, it is not allowed). Microsoft complaining about the problems that keep them from embracing software released under the GPL is as silly as a free software author complaining about the fact that he or she cannot incorporate the source from Microsoft Word because it would violate both Microsoft's copyright and the license under which it distributes its product.

In short, code licensed under the GPL can be used for most purposes without worry, and those who do need to worry have an explicit choice to make. Either they can use the software and abide by its terms (which are admittedly viral in nature), or they can find a solution that is not dependant on such software.

Myth #7: The GPL forces you to release your proprietary changes.

This myth is probably the most widespread misconception about the way the GPL works. The Alexis de Tocqueville report uses this myth to claim that the government could not use software licensed under the GPL in anything that has national security implications, because the government would be obliged to release sensitive information. As we discussed in Myth #5, the government certainly need not worry if they are only end-users of such software.

In cases where the government makes classified changes to software licensed under the GPL, they are still protected. This is because the GPL does not actually force any entity to distribute changes made to software using that license. Instead, it states that if you do distribute the software, then the entity or entities receiving the software must be able to redistribute it freely if they so desire.

Clearly, if an entity such as a government does not wish to see its changes distributed, then it need not distribute those changes.

In the case of the United States government, it can legally prevent further distribution of its own changes by classifying the work. Remember that the protection the GPL offers is solely rooted in the protections afforded by copyright law. The government can deny an otherwise right to copy in the interests of national security.

According to Eben Moglen, the General Counsel of the Free Software Foundation, “Software that performs functions that it is a matter of national security to keep secret should be classified, and there is nothing in the GPL that can thwart the classification system”.

Myth #8: *Free software destroys claims to “intellectual property”.*

Many people are afraid that their copyrights, patents and trademarks may be at risk when producing free software. Generally, that is not the case. First, trademarks are a completely independent issue that does not impact the free software developer. Let’s say that Microsoft were to provide a version of Microsoft Word licensed under the GPL. The GPL does not in any way prevent Microsoft for pursuing trademark violations of the “Microsoft Word” name in any derived work. No other free software license makes such restrictions either.

In terms of patent protection, it is true that giving away an implementation of a patented algorithm might have a negative effect on revenue derived from that patent. However, it in no way nullifies the patent. If a company wished to give away a product that uses their patent under the GPL, then they could simply grant a royalty free license to anyone using a work derived from their original software release. What’s more, any competitors would not be allowed to use the implementation of the patent in their own system without complying with the terms of the GPL. If those competitors want to use the patented algorithm without placing their software under the GPL, then they would be forced to license the patent.

Additionally, one may separate patented algorithms into a separate piece of software that is licensed differently. The two pieces of software could interoperate through any means that does not violate the terms of the free software license.

Finally, some people seem to believe that one is giving up copyright on a piece of software by releasing it under a free software license. Such a belief signals a fundamental misunderstanding of the way copyrights work. First, in the US and many other countries, any creative work is automatically copyrighted. There is no need to register the copyright, though registration can make a copyright easier to defend. Second, giving people permission to copy cannot revoke the copyright. The idea behind copyright is that the entity that creates a work should have ultimate say over how the work may be copied. Copyright gives the authoring entity the right to enter into contracts that may limit or transfer the right to copy. Indeed, the average free software license gives nearly unlimited permission to copy. However, the actual copyright still exists, and when it finally expires, then the work will be in the public domain. As a result, any restrictions that a license does impose will generally become irrelevant.

Myth #9: *The GPL would not stand up in court (or, it is unclear whether it would).*

Unlike many software licenses, the GPL is trivially enforceable, at least throughout the United States, because the legal principles behind the license are rooted in a well-interpreted area: copyright law.

The GPL derives every last bit of its power from copyright law. The license allows you to receive a copy of any piece of software licensed under it, but forbids you from making your own copies of that software unless you comply with the terms of the license. If you copy the software in such a way that the license is violated, then you are violating copyright law. It is that simple, because the laws governing copyright are relatively simple and clear, and they are designed to protect the interests of the copyright holder.

If Stephen King were to write a new book, and a publisher made and distributed copies outside the bounds of their agreement, he would surely win in a court of law. There are cases where the author does not have complete control over copies. For example, under copyright law, you may take an album you own and make a personal copy of it, and even loan that copy to a friend. However, such use of the work is the kind of use that the free software movement promotes, so it is not an issue in this context.

Eben Moglen, the Free Software Foundation's General Counsel, has written two informative pieces about the enforceability of the GPL, both available from his web page at <http://emoglen.law.columbia.edu/>. In his articles, he explains that he has been able to enforce the GPL in "dozens" of instances, and has not ever had to step foot into court, because the GPL is really that straightforward to enforce. Essentially, for those critics who claim that a court case is necessary to prove the worth of the GPL, Eben argues strongly that the number of times he's been able to enforce it without going to court clearly demonstrates that no one has yet been foolhardy enough to go to court over this issue.

Myth #10: *Using free software in a project implies that the project has no path to revenue.*

In Myth #5, we saw that one can be an end-user of free software and make money without problem. Companies like Red Hat make good money selling free software (most of which they did not write), and many small but successful ISPs run on nothing but free operating systems. Here, we will focus our attention on using free software in an actual software product.

This myth is often stated more vaguely using the GPL as an example by saying, "The GPL is an infeasible business model". The GPL is not, in and of itself, a business model; it is only a license. We assume that critics believe that it is difficult to impossible to realize revenues when free software (particularly that licensed under the GPL) is involved.

True, it looks pretty dismal for most people who build their business model around selling software, because why would someone buy software they could get for free?

One answer to the above question is support. There are plenty of companies that make money by providing support for free software. Similarly, there are a few companies that give away “loss-leader” (crippled) versions of their software or services under a free software license in order to drum up interest and perhaps build up credibility, so that people will hopefully buy the product that actually makes them money.

Other organizations such as IBM simply use free software for internal reasons in the course of doing business, where there would be no monetary harm in distributing their changes. In such cases, the free software is not directly related to the bottom line of their business, but releasing that software can create goodwill, and could potentially enable enhancements or other works that do add value in an indirect manner.

There are two key areas we see where the GPL in particular can occasionally play an important part in an effective business model. The first is for those software shops that are selling libraries that are aimed at developers. Releasing such software under the GPL can bring publicity and goodwill. Generally, the people who use the released library in a free environment were unlikely ever to pay for the library. However, anyone wishing to use the software in a non-free environment would need to negotiate a different license at a fee.

The second area where we see the GPL supporting a business model is for companies where software is not the main business of the company. In particular, consider hardware manufacturers. By releasing their drivers as free software, they may see a competitive advantage. In particular, making their drivers available can lead to rapid adoption of their hardware in the Unix world, because the free software community would not need to expend the time and energy to reverse engineer a Windows driver (as often happens). Manufacturer-supported drivers tend to be more robust than something built by reverse engineering a proprietary driver, providing the end user another reason to choose that vendor’s product besides the ability for early adoption.

Unfortunately, there is not much factual evidence to support whether good business models can be built around free software as a core component. There are anecdotal successes and failures, but there aren’t enough data points to make a solid determination.

However, there are obviously many cases where the license of a particular piece of software is irrelevant to the business model.

We also admit that a good business model revolving around free software must find significant revenue streams that aren’t related to selling the software, and in some markets, such streams may not be feasible. In such situations, simply do not release your product as free software.

Myth #11: Free software vendors do not offer warranties or accept liability.

Most consumer software licenses for commercial products do not offer any warranties as to the behavior of software, nor do the companies offering the software generally accept any liability. In many cases, these points can be negotiated for an additional fee.

Why should the free software community be any different? It doesn't make economic sense for any software vendor, from Microsoft to the Dan Bernstein, to offer warranties or accept liability. However, just as many companies would be willing to reconsider for the right remuneration, so would be most development teams on free software projects.

Myth #12: *You can't get support for free software.*

The free software community is largely staffed by part-time volunteers, who are often too busy to support their projects as well as they should be. However, many such developers would be more prone to find time if they were being paid for their work, even if that work were merely in support of their project. Additionally, there are companies that specialize in finding developers to meet the free software needs of companies, such as CollabNet.

Myth #13: *Free software development is an anarchic mess that leads to shoddy code.*

We have never seen any well-supported factual basis to support this myth. In fact, the popular opinion among developers as well as a wealth of anecdotal evidence suggests that the opposite may be true (i.e., free software projects tend to be more robust than proprietary projects).

Free software projects can have many contributors, but few projects have more than three that contribute regularly. Bigger projects can have many more regular contributors, but will always have some sort of effective management structure.

The key argument that most developers believe is the key reason why free software tends to be more robust is the "many eyeballs" phenomenon, which states that source availability leads to more people looking at code, which leads to more people finding and resolving bugs.

The anecdotal evidence is somewhat compelling. However, we believe that the jury is still out on this issue.

Myth #14: *Free software is intrinsically more difficult to use.*

Yes, as the de Tocqueville report suggests, our mothers probably could not use Debian GNU/Linux. However, the lack of usability in most free software is due to the fact that our grandmothers are not the target market of the free software community. That community currently tends to target IT professionals, network engineers and hobbyists. That is, it is largely self-serving for the moment, though there are some projects that are focused on usability.

The truth is that there is nothing intrinsic about free software that makes it more difficult to use. The status quo was brought about because the market rewards proprietary software for usability, whereas free software authors tend to see no reward for it.

One interesting point is that there is plenty of room for the commercial sector to add usability features on top of free software. For the most part, this is exactly what Macintosh is doing with Mac OS X. That operating system puts wrappers around several pieces of prominent free software, and combines it with other proprietary software. And, in fact, one of our mothers does use this operating system, and is quite happy with it.

Conclusion

The report “Opening the Open Source Debate” generally serves to propagate all of the myths we have discussed in this paper, many of which others have argued in the past. The common thread among all these critics and all their arguments seems to be that software companies and governments should avoid free software, particularly anything licensed under the GPL.

In this paper, we’ve looked at a number of arguments such critics commonly use in attempt to dissuade those who might think of using free software in their endeavors. In each case, the argument is based on misconceptions. First, most arguments are really directed at software released under the GPL instead of free software in general. Second, arguments involving the GPL are nearly always based on a fundamental misunderstanding of the way the GPL works.

In reality, any organization that wishes to use a piece of software should carefully consider the licensing issues. Yes, the GPL has implications that a company might not like. In particular, derived works must retain the terms of the original license, and there are certainly many cases where that restriction makes software completely unsuitable to the task.

However, the license that comes with any piece of proprietary software may have implications that are just as bad. In the end, organizations should be making informed decisions on what off-the-shelf software may work for them based on the facts, not on fear, uncertainty and doubt spread by lobbyists from either camp.

About the Authors

John Viega is a Senior Research Scientist at the Cyberspace Policy Institute (www.cpi.seas.gwu.edu), the CTO of Secure Software, Inc. (www.securesw.com), and an Adjunct Professor of Computer Science at Virginia Tech. He is co-author of the books *Building Secure Software*, *Network Security with OpenSSL*, and the forthcoming books, *The Secure Programming Cookbook* and *The Network Security Cookbook*.

Bob Fleck is a Research Scientist at the Cyberspace Policy Institute, the Director of Methodology Development at Secure Software, Inc., and is the co-author of the book *Deploying Secure Wireless Networks*, due out in September from O’Reilly and Associates. He has been quoted on security-related issues in major media publications such as CNN and the Wall Street Journal.