Distributed Software Development via Freenet

Peter Conrad

Table of Contents

License

Overview

Free Software and developers of free software are under attack. This is nothing new. However, during recent years the attacks have become heavier as well as more common. It can be anticipated that this trend is not going to change in the foreseeable future. In fact, certain political developments indicate that the problems for software developers are getting worse.

Some developers may find it desirable to not have their names mentioned in connection with certain software projects. So far, the only reliable means to prevent that is to simply not participate in the development of free and open source software. This can mean that in the future either

- people won’t create patches to existing free software, or
- people will create new software or patches to existing software but will not publish it.

Obviously, both of these options are very bad for the free software movement, which is critically reliant on active participation of users and developers. Therefore, it seems desirable to present a third option: secure anonymous distributed software development.

This paper is going to discuss the dangers that software developers have been and still are facing, and it is going to introduce a possible means for secure anonymous distributed software development.

The Dangers of Software Development

Different kinds of attacks on free software and its developers exist. This paper is not about FUD (fear, uncertainty and doubt) attacks usually employed by large software companies. Those usually are of a more general kind and pose no immediate threat to the developers themselves.

Instead, it is going to focus on two other attack methods: criminalization and civil lawsuits.

Criminalization of Developers

If you ever started a free software project, or if you ever submitted a patch for another project, you probably have never worried about being put into jail for it. At least I didn't worry when I wrote a ZIP password cracking [http://www.unix-ag.uni-kl.de/~conrad/krypto/pkcrack.html] program in 1995. But today I am.

Believe it or not, there are numerous examples of programmers ending up in jail basically for writing software, even for software where it isn't immediately obvious that such a thing could be possible.

A famous (the most famous?) example for a criminalization attack is the author of PGP [http://www.pgpi.org/] ("Pretty Good Privacy"), Philip Zimmermann [http://www.philzimmermann.com/]. PGP is a program for encrypting files or email messages in a very secure way. While writing and / or using encryption software at that time was perfectly legal (in the US), exporting such software was not. Although Phil never actually went to jail, he has been under investigation for violating export control regulations for three years. He had to spend a huge amount of money on his defense team and was facing a prison sentence until the case was dropped.

A more recent example where a programmer actually went to jail is the case of Dimitry Skylarov [http://www.vnunet.com/News/1125109], a Russian programmer. For his employer ELCOMSoft [http://www.elcomsoft.com/], a software company situated in Russia, he wrote a program for breaking into Adobe's [http://www.adobe.com/] ebook format. When he attended the DefCon hacker convention in Las Vegas in 2001, he was arrested for violation of the "Digital Millenium Copyright Act" (DMCA). Although he was never actually found guilty, he spent about three weeks in prison before he was released. The case against him was only dropped after he agreed to be a witness against his employer in another trial.

A case that is still unresolved is the quite famous "DeCSS case" [http://www.theregister.co.uk/content/archive/23633.html] of Jon Johanssen of Norway. His "crime" was to write a program so he could view his legally bought DVD movies on his Linux computer. Although he is probably not facing a prison sentence because at the time he wrote the program he was only 15 years old, his case will continue later this year and is likely to continue causing him trouble.

The point of all these examples is that you don't have to do anything that's actually illegal to get into trouble. If Dimitry Skylarov had anticipated that he would be charged for writing a program in Russia that is legal in Russia he probably wouldn't have gone to the DefCon meeting. Who knows if today we would have software like PGP if Phil Zimmermann had known about the investigation that was to be conducted against him?

Civil Lawsuits

Civil lawsuits against programmers are usually based on patent / trademark or even copyright issues. While these usually do not endanger the programmer of being jailed, they can pose a serious economic risk. Luckily, such actions against individual programmers are not very common, mostly because the financial gain for the plaintiff from winning a lawsuit against an individual programmer is comparatively small.

However, unless some kind of mutual agreement by the parties is found, such lawsuits can seriously hamper progress of a free software project, and even completely extinguish it.

Once again, PGP is a nice example. Phil Zimmermann managed to violate two patents in one program: the patent on the RSA algorithm held until a few years ago by RSA Security, Inc. [http://www.rsa.com/], and the IDEA patent held by Ascom AG [http://www.ascom.ch/] of Switzerland. Although an agreement was found so that PGP could be used freely for non-commercial purposes, these patents are probably the main reason why PGP was practically non-existent in the commercial world for a very long time. Moreover, the RSA agreement meant that two different versions of the RSA code had to be maintained.

Sony Corporation [http://www.sony.com/] have been producing their robot dog "Aibo [http://www.aibo.com/]" for a couple of years now. An owner of that robot was less than satisfied with the software that Sony offered for the robot, and therefore freely distributed his own improvements to Sony's software to other Aibo owners. Despite the fact that the sales of the "dog" and Sony memory sticks benefited from his software (which was even acknowledged by Sony at some time), he was finally bullied by Sony [http://www.wired.com/news/business/0,1367,48088,00.html] to take the software off the internet.

A less dramatic example is the Unisys GIF patent [http://cloanto.com/users/mcb/19950127giflzw.html]. Actually, GIF images as such are not covered by the patent, but the algorithm used for compression in GIF images is. After GIF images became quite popular, in the mid-'90s Unisys decided that a license was required for anyone using software for creating compressed GIF images. As a consequence, free software for creating compressed GIF images practically vanished. The giflib [http://prtr-13.ucsc.edu/~badger/software/libungif/giflib.shtml] library, at that time maintained by well-known Eric S. Raymond, is still available today, but was last modified in 1999. The positive side of this is that thereby Unisys enforced the specification of a much better alternative, i. e. the PNG image format.

An extremely silly example is what happened to Tom Lord, maintainer of the Arch revision control system. He was asked (politely) by BitMover, Inc. to remove the phrase "Line of Development" from the Arch documentation because apparently BitMover own a trademark on that phrase. That trademark is highly questionable, because "Line of Development" is a commonly used term in revision control (e. g. the CVS FAQ mentions it as well). It is worth noting that despite the silliness of the trademark they could've sued Tom Lord instead of just asking.

These examples demonstrate that patent issues in particular can form a severe impediment of the development of any kind of software. While the question of software patents is still largely unresolved in Europe, it seems reasonable to assume that the problems will be getting worse in the future.

Conclusions

The given examples vividly display the threats that programmers are facing today. Laws like the DMCA are being passed in other countries as well. E. g. in Germany a similar law was passed in April 2003. At the same time, the industry is making moves of taking away freedoms of computer users, e. g. with "copy-protected" audio disks or the TCPA aka Palladium aka NGSC efforts. Programmers could re-instate some of these freedoms, like Jon Johanssen tried to, but in the face of the aforementioned threats it is questionable how many programmers will actually take that risk.

The conclusion for me is that I am not going to enter the United States in the foreseeable future, because my ZIP cracking program is probably also a violation of the DMCA, and I have no intent of becoming the next Skylarov.

A way out

A possible way out of this dilemma has already been mentioned: secure anonymous distributed software development.

More specifically, what I'm proposing here is pseudonymous distributed software development using freenet.

"Pseudonymous" means that developers can create a securely identifiable pseudonym for themselves without the danger of conveying any personally identifiable information. This is actually quite important, not just for distributed software development, but also for any other form of cooperation. The reason for this is the concept of "trust": work is much easier if you can trust the people you work with. To build trust it is not required to know

each other personally. It is sufficient to be able to reliably identify cooperators. Trust then builds up by evaluating the contributions of reliably identified individuals.

When you think about it, the concept of trust based on pseudonyms is quite common. For example, very few people know Linus Torvalds personally. Yet millions of people trust Linus to create useful operating system software. Not because of personal relations to him, but simply because his contributions to the linux kernel and the community as a whole have been remarkably useful. Therefore it makes sense to assume that the next release of the linux kernel will be useful as well.

Taking this to extremes means that it doesn't even matter if Linus is an individual working for Transmeta or if "Linus" is actually a special task force working for Microsoft Corporation. The important point is that whoever (or whatever) "Linus" is, "Linus" has contributed well and therefore will probably be contributing well in the future.

We only have to make sure that neither Transmeta nor Microsoft Corporation manage to achieve World Domination. :-)

Freenet

Freenet [http://freenet.sf.net/] is a Peer-to-Peer (P2P) network with a strong focus on anonymity and resistance against any form of censorship. In contrast to many other popular P2P networks, freenet is not a filesharing network in the usual sense. Instead of just "sharing a folder", content has to be inserted into freenet explicitly. Therefore, it makes sense to view freenet as a large, unreliable, online storage media.

The freenet network is made up of individual "nodes". Nodes communicate with each other over encrypted connections, exchanging files and other information, like e. g. routing paths. Clients connect to individual nodes to retrieve files from the network. Ususally, nodes restrict client access to very few IP numbers, or even localhost only. There are some publicly accessible nodes, though, like e. g. this one [http://68.38.52.138:8888/]. If you use a public node for client communication, you should be aware that the communication between your client and the node is *not* secure! In other words, anyone who wants to seriously use freenet should run his own node.

If you have tried the previous link to the public node you'll have noticed that an ordinary web browser can be a freenet client. In this case, an HTTP proxy server is connecting the browser to freenet. People have created complete websites in freenet (so-called "freesites"), e. g. expressing political opinions they'd probably not express on a traceable web server. Other types of clients can connect to the node using a different protocol, which is mostly meant for programmatic access.

Content in freenet is accessed (or uploaded) using "keys". There are three types of keys:

- Content Hash Keys (CHK) are basically a secure hash of the file that is accessible under that key.
- Secure Subspace Keys (SSK) basically consist of an identifier string prefixed with a public key. The content is signed with that key, which means that you need the corresponding private key to insert the content.

  Because of this, the public key can be viewed as a kind of pseudonym. Although the owner of the private key can give the private key to someone else, it is impossible to impersonate him without knowledge of the private key, which should be difficult to get without his consent.
- Keyword Hash Keys (KSK) are basically just some identifier string. This key type is *not* secure, because anyone can insert any content under any given KSK. It is even possible (and has been done) to "hijack" existing KSK keys.

  However, KSK keys are useful for two-way communications. By simply introducing a naming convention like "message-1.txt", "message-2.txt" etc. people can post and retrieve messages without having to exchange SSK key information *before* the actual communication.

Whenever someone retrieves content from the network, the content is replicated to all the nodes that happen to be in the search path. This is one of the reasons why freenet is pretty resistant against censorship: taking down individual nodes will likely not take down some specific content. The mere process of checking if certain content is available in freenet is causing the propagation of same content!

This also has a drawback: you cannot re-use keys. (Actually you can, as has been shown with the hijacking mentioned above, but the process is unreliable and cumbersome.) To overcome this problem, two tricks are being used:

- Date Based Redirects (DBR): instead of inserting content under a given key "A", some "meta information" is inserted, which basically instructs the client to calculate a timestamp value and issue another query for "<timestamp>-A". The drawback of this method is that the maintainer of the content under key "A" has to make sure the content is regularly re-inserted under "<timestamp>-A" for appropriate timestamps.
- Editions: edition-based keys use a similar naming convention as mentioned for KSK communications above. Instead of inserting content under key "A", it gets inserted under "A/1". Whenever the maintainer wants to update the content, he inserts the new content under "A/2", increasing the edition number with each update. For convenience, "A/1" usually contains a link to the succeeding edition. The drawback of this method is that the current edition number must be communicated somehow.

It is possible to combine both methods by using a DBR key to redirect the client to the current edition.

Arch

Arch [http://arch.fifthvision.net/] is a revision control system designed and written (mostly) by Tom Lord. An important feature of arch is that it is distributed by design. Specifically, it allows every developer to have his own repository. Branching and merging across repository boundaries are possible. Support for locally caching remote archives or parts thereof is also included, which can significantly speed up operations on remote repositories. And it already has a simple API for accessing remote repositories over different network protocols like http, WebDAV, ftp and even sftp (ftp over ssh).

All of these features make arch the preferred choice for software development via freenet: it can easily be adapted to access in-freenet repositories. Every developer can maintain his own repository in freenet, which means every developer can publish his archive under his own SSK. This, in turn, is important for creating trust as described above.

Another good thing about arch repositories is that the files inside the repository hardly ever change. Committing changes into the repository essentially creates a few (less than 5) files inside a new directory in the repository tree, with the bulk being contained in a .tar.gz file. This means that updating an in-freenet repository only requires the upload of the new files, as opposed to uploading possibly many or large files, as would be required by other popular revision control systems.

For people used to CVS-like systems, migration to arch can be a little confusing or even irritating. That is not a result of bad design, though, but rather a result of a quite different design, and in some parts a result of different wording in the respective documentation and tools. However, there is a very good tutorial available [http://regexps.srparish.net/tutorial/arch.html].

Throwing them together

The required patches to arch for accessing freenet are very small and touch only very few places, which speaks for good software design in arch. The areas concerned are:

- with-ftp, the general utility for accessing multiple types of remote repositories over a common interface. It has been extended to allow freenet: type URIs
- push-mirror has been extended for publishing an archive into freenet, by introducing a similar command publish-freenet-mirror

- an additional small utility for anonymizing tar archives has been added (tar archives store user and group information, which compromises the developers' anonymity!)

- other small helper scripts and programs have been added as well

For efficiency, additional files have been placed in the archive's "=meta-info" directory. All of these start with "freenet-" to distinguish them from the rest of the meta-info files. These are not mirrored into freenet.

The freenet-enabled version of arch can be built by checking out the patched version from the arch repository at http://www.unix-ag.uni-kl.de/~conrad/Archives/DSDiF/ or from freenet itself at freenet:SSK@LCgWj0qabAqxUNKbfI93PCTH9RUPAgM/DSDiF [freenet:SSK@LCgWj0qabAqxUNKbfI93PCTH9RUPAgM/DS or by downloading the prepared source package from the same URL. It can be compiled, installed and used just like the unpatched version of arch, but please take note of the section "Recommended Practice" below if you're interested in preserving your anonymity.

In order to access archives in freenet, you must simply register the freenet: URI with the command "larch register-archive". Then you can access the in-freenet archive just like any other remote archive with the usual arch commands. Publishing your own archive into freenet is done using the command "larch push-mirror", similar to the way "normal" mirrors can be created.

Does it work?

Well, kind of. At this time I wouldn't start moving a large project like the linux kernel into freenet.

Technically, it does work. Although real-life demo projects don't exist yet, as a proof of concept the arch patches themselves have been mirrored into freenet, and I have managed (using the patched version, of course) to successfully build another patched arch by checking out code from the main distribution site and the patches in freenet.

The biggest problem with the proposed solution at this time is speed. Arch without freenet is already slow compared to CVS, although great improvements are being made in this respect. Freenet is very slow. It is hardly usable over a slow modem link, and it is still slow when you set up a permanently running node on a broadband connection. The combination of both arch and freenet is *extremely* slow.

The second biggest problem at this time is probably reliability. Freenet is (by design) not a permanent storage media. Freenet promotes popular content by replicating it, which means that unpopular content will sooner or later drop out of the network. Therefore, to improve availability of your repository, you must re-insert it at regular intervals. Which, again, can be very slow.

However, while these problems are not making life easier, they also aren't showstopper problems. It has already been mentioned that arch allows local caching of remote repositories, which should be made ample use of. The process of inserting or re-inserting an archive into freenet can run asynchronously in the background without interrupting ongoing work. The retrieval and caching of fellow developers' repositories can also run asynchronously in the background. So eventually other people's patches will end up in the local cache, from where they can quickly be integrated into the local development branch.

Probably the only thing that's not possible with this setup is an extremely rapid development cycle. That's not necessarily bad, though - quite a few projects could benefit from a slow think-before-you-code pace... ;-)

Recommended Practice - Anonymity

If you want to develop a software project via Freenet, it is not unlikely that you do so because Freenet offers anonymity. Anonymous development is quite uncommon in Open Source and Free Software Projects, though, and therefore revision control software is usually trying pretty hard to make sure you receive credit for your contributions. So you should take precautions to set up a development environment that makes sure arch does not include any hint on your identity into the project repository.

First, you should set up an encrypted filesystem, e. g. using loopback encryption on a linux machine. Make sure it is mounted under some unconspicuous name like "/encrypted". (Using e. g. "/home/torvalds/cryptofs" would

probably be a bad idea if your name was Linus Torvalds.) Local encryption is not required for insertion into freenet, it is meant to protect you when some guys wearing sunglasses take away your beloved little computer in a black helicopter.

Second, create an unconspicuous "development home" directory on the encrypted filesystem, like e. g. "/encrypted/dev/CoolProject".

Third, change your $HOME environment variable to point to that directory before setting up your arch environment and the rest of the project. This is a little tricky, because if you do this manually in your shell the commands may end up in your shell's history file! Therefore, create a small helper shell script (with an unconspicuous name, of course, like "/encrypted/script1") that contains something like this:

```
#!/bin/sh

HOME=/encrypted/dev/CoolProject
export HOME
cd $HOME
$SHELL -i
```

Only after executing that script should you initialize your arch environment and work on the project. Arch saves configuration information in "$HOME/.arch-params", which will automatically end up in the encrypted filesystem. Of course, you should create your "{archive}" and "{repository}" directories on the encrypted volume, too. It might be clever to use a different "{archive}" for each project, so you can publish them independently. And don't use your email address as your Arch user ID! :-)

Conclusion

As has been shown by example, programmers take a certain risk when developing software. This is also true for development of free and open source software.

With the combination of revision control software and a secure anonymous file storage media, a technical means is now available to reduce that risk significantly. The price the programmer has to pay, though, is high as well, because he can no longer take personal credit for his work. This can be a problem, because due credit is an important motivating factor in the free software community.

In any case, the problems programmers are facing are of a political nature. A good solution for these problems must neccessarily be a political solution. What has been presented in this paper can therefore not be more than a technical workaround to these political problems. It remains to be seen if the workaround is at least an effective one.

Disclaimer

IANALAIDPOOTV - I Am Not A Lawyer And I Don't Play One On TV