

# Source Disclosure is Not The End

*a rebuttal of spurious arguments  
against disclosed source for  
electoral software*

Josh Berkus, 2006

# Why Me?

*Josh Berkus is a famous Open Source software expert.*

- Has 11 years programming experience on database systems.
- Spent over four years as on the PostgreSQL (Open Source database software with over 15 million users) Core Team
- Works for Sun Microsystems helping them with Open Source strategy.\*
- Was a project lead for OpenOffice.org (open source office suite with over 60 million users) for two years.
- Has contributed to numerous other projects including OpenDocument, Bricolage, and GForge
- Non-voting board member of Software in the Public Interest
- Speaker at technology conferences around the world, including San Francisco, New York, Canada, Brazil, Germany and Japan.

*(\* = please note that Josh Berkus is not here representing Sun Microsystems in any way. He is here strictly as a concerned citizen and a software expert)*

# How is Open Source Relevant?

*The proposed bill would not require Open Source software to be used for voting systems, although it would allow qualifying Open Source software to be used. Instead, it merely requires that voting source software manufacturers disclose their source code, without giving up any commercial rights, to the public to ensure that everyone's vote is counted correctly. This is called "Disclosed Source".*

*The reason why Open Source software is relevant to this proposal is that the vendors opposing the bill are using the exact same arguments which have been used, unsuccessfully, to oppose the advance of Open Source software.*

# The Security Argument

*“Disclosing our source would make our voting software less secure.”*

## **Secrets *decrease* security, openness *increases* it:**

- The more expert eyes see the code, the more bugs which will be found – and fixed.
- “A secret that cannot be readily changed should be regarded as a vulnerability.” (see attached article)
- Attackers do not need access to the source code to succeed in attacks (see attached article).
- Vincent Rijmen, a developer of the winning Advanced Encryption Standard (AES) encryption algorithm used by the US government says of the open code operating system Linux: *“the open source nature of Linux provides a superior vehicle to making security vulnerabilities easier to spot and fix, not only because more people can look at it, but, more importantly, because the model forces people to write more clear code, and to adhere to standards. This in turn facilitates security review.”*

## **PostgreSQL as security example**

- PostgreSQL's source code has been available on the Internet for 10 years.
- The PostgreSQL database management system is used by over 15 million people worldwide in a variety of industries.
- PostgreSQL is used within the US defense industry by several companies, agencies and departments.
- PostgreSQL has not had a single known exploit in the field in at least six years.
- The Database Hacker's Handbook (2005), comparing PostgreSQL to Oracle, MS SQL Server, DB2, Informix and MySQL evaluated PostgreSQL as:

*... by default, PostgreSQL is possibly the most security-aware database available ...*

That means *more* secure than Oracle.

# The Business Argument

*“If we disclose our source, we will not be able to make money, and will have to leave the state.”*

## No Giveaway

- Vendors are not being asked to give up their copyrights or patents. Which means that they can continue to charge their customary fees for software.
- Copyright, without secrecy, is enough for the publishing industry; why would it not work for voting software?

## Open Source is Profitable

- Hundreds of companies in California and the rest of the world are investing in Open Source software; clearly they think they can make money from it:

Red Hat	Sun Microsystems	IBM
Novell	Oracle	Google
Nokia	Fujitsu	Cisco
NTT	Yahoo	HP

- One could even argue that California's recent economic recovery is fueled by Open Source software, which is at the heart of dozens of new Silicon Valley start-ups. (see attached article)

# From Secure Programming for Linux and Unix

David Wheeler (2000)

## 2.4.2. Why Closing the Source Doesn't Halt Attacks

It's been argued that a system without source code is more secure because, since there's less information available for an attacker, it should be harder for an attacker to find the vulnerabilities. This argument has a number of weaknesses, however, because although source code is extremely important when trying to add new capabilities to a program, attackers generally don't need source code to find a vulnerability.

First, it's important to distinguish between "destructive" acts and "constructive" acts. In the real world, it is much easier to destroy a car than to build one. In the software world, it is much easier to find and exploit a vulnerability than to add new significant new functionality to that software. Attackers have many advantages against defenders because of this difference. Software developers must try to have no security-relevant mistakes anywhere in their code, while attackers only need to find one. Developers are primarily paid to get their programs to work... attackers don't need to make the program work, they only need to find a single weakness. And as I'll describe in a moment, it takes less information to attack a program than to modify one.

Generally attackers (against both open and closed programs) start by knowing about the general kinds of security problems programs have. There's no point in hiding this information; it's already out, and in any case, defenders need that kind of information to defend themselves. Attackers then use techniques to try to find those problems; I'll group the techniques into "dynamic" techniques (where you run the program) and "static" techniques (where you examine the program's code - be it source code or machine code).

In "dynamic" approaches, an attacker runs the program, sending it data (often problematic data), and sees if the programs' response indicates a common vulnerability. Open and closed programs have no difference here, since the attacker isn't looking at code. Attackers may also look at the code, the "static" approach. For open source software, they'll probably look at the source code and search it for patterns. For closed source software, they might search the machine code (usually presented in assembly language format to simplify the task) for essentially the same patterns. They might also use tools called "decompilers" that turn the machine code back into source code and then search the source code for the vulnerable patterns (the same way they would search for vulnerabilities in open source software). See Flake [2001] for one discussion of how closed code can still be examined for security vulnerabilities (e.g., using disassemblers). This point is important: even if an attacker wanted to use source code to find a vulnerability, a closed source program has no advantage, because the attacker can use a disassembler to re-create the source code of the product.

Non-developers might ask "if decompilers can create source code from machine code, then why do developers say they need source code instead of just machine code?" The problem is that although developers don't need source code to find security problems, developers do need source code to make substantial improvements to the program. Although decompilers can turn machine code back into a "source code" of sorts, the resulting source code is extremely hard to modify. Typically most understandable names are lost, so instead of variables like "grand\_total" you get "x123123", instead of methods like "display\_warning" you get "f123124", and the code itself may have splatterings of assembly in it. Also, `_ALL_` comments and design information are lost. This isn't a serious problem for finding security problems, because generally you're searching for patterns indicating vulnerabilities, not for internal variable or method names. Thus, decompilers can be useful for finding ways to attack programs, but aren't helpful for updating programs.

Thus, developers will say "source code is vital" when they intend to add functionality), but the fact that the source code for closed source programs is hidden doesn't protect the program very much.

### **2.4.3. Why Keeping Vulnerabilities Secret Doesn't Make Them Go Away**

Sometimes it's noted that a vulnerability that exists but is unknown can't be exploited, so the system "practically secure." In theory this is true, but the problem is that once someone finds the vulnerability, the finder may just exploit the vulnerability instead of helping to fix it. Having unknown vulnerabilities doesn't really make the vulnerabilities go away; it simply means that the vulnerabilities are a time bomb, with no way to know when they'll be exploited. Fundamentally, the problem of someone exploiting a vulnerability they discover is a problem for both open and closed source systems.

One related claim sometimes made (though not as directly related to OSS/FS) is that people should not post warnings about vulnerabilities and discuss them. This sounds good in theory, but the problem is that attackers already distribute information about vulnerabilities through a large number of channels. In short, such approaches would leave defenders vulnerable, while doing nothing to inhibit attackers. In the past, companies actively tried to prevent disclosure of vulnerabilities, but experience showed that, in general, companies didn't fix vulnerabilities until they were widely known to their users (who could then insist that the vulnerabilities be fixed). This is all part of the argument for "full disclosure." Gartner Group has a blunt commentary in a CNET.com article titled "Commentary: Hype is the real issue - Tech News." They stated:

The comments of Microsoft's Scott Culp, manager of the company's security response center, echo a common refrain in a long, ongoing battle over information. Discussions of morality regarding the distribution of information go way back and are very familiar. Several centuries ago, for example, the church tried to squelch Copernicus' and Galileo's theory of the sun being at the center of the solar system... Culp's attempt to blame "information security professionals" for the recent spate of vulnerabilities in Microsoft products is at best disingenuous. Perhaps, it also represents an attempt to deflect criticism from the company that built those products... [The] efforts of all parties contribute to a continuous process of improvement. The more widely vulnerabilities become known, the more quickly they get fixed.

### **2.4.4. How OSS/FS Counters Trojan Horses**

It's sometimes argued that open source programs, because there's no enforced control by a single company, permit people to insert Trojan Horses and other malicious code. Trojan horses can be inserted into open source code, true, but they can also be inserted into proprietary code. A disgruntled or bribed employee can insert malicious code, and in many organizations it's much less likely to be found than in an open source program. After all, no one outside the organization can review the source code, and few companies review their code internally (or, even if they do, few can be assured that the reviewed code is actually what is used). And the notion that a closed-source company can be sued later has little evidence; nearly all licenses disclaim all warranties, and courts have generally not held software development companies liable.

Borland's InterBase server is an interesting case in point. Some time between 1992 and 1994, Borland inserted an intentional "back door" into their database server, "InterBase". This back door allowed any local or remote user to manipulate any database object and install arbitrary programs, and in some cases could lead to controlling the machine as "root". This vulnerability stayed in the product for at

least 6 years - no one else could review the product, and Borland had no incentive to remove the vulnerability. Then Borland released its source code on July 2000. The "Firebird" project began working with the source code, and uncovered this serious security problem with InterBase in December 2000. By January 2001 the CERT announced the existence of this back door as [CERT advisory CA-2001-01](#). What's discouraging is that the backdoor can be easily found simply by looking at an ASCII dump of the program (a common cracker trick). Once this problem was found by open source developers reviewing the code, it was patched quickly. You could argue that, by keeping the password unknown, the program stayed safe, and that opening the source made the program less secure. I think this is nonsense, since ASCII dumps are trivial to do and well-known as a standard attack technique, and not all attackers have sudden urges to announce vulnerabilities - in fact, there's no way to be certain that this vulnerability has not been exploited many times. It's clear that after the source was opened, the source code was reviewed over time, and the vulnerabilities found and fixed. One way to characterize this is to say that the original code was vulnerable, its vulnerabilities became easier to exploit when it was first made open source, and then finally these vulnerabilities were fixed.

### **2.4.5. Other Advantages**

The advantages of having source code open extends not just to software that is being attacked, but also extends to vulnerability assessment scanners. Vulnerability assessment scanners intentionally look for vulnerabilities in configured systems. A recent Network Computing evaluation found that the best scanner (which, among other things, found the most legitimate vulnerabilities) was Nessus, an open source scanner [Forristal 2001].