

Will openish source really improve security?

Gary McGraw
Reliable Software Technologies
Dulles, VA 20166
<http://www.rstcorp.com>
gem@rstcorp.com

Abstract

Will openish source really improve security? No.

1. Introduction

I am using the term “openish source” as a reaction to the fact that the OSI has hijacked the term “open source” and the natural definition most people likely intuit does not apply. The term I am using is ridiculous. I chose an intentionally ridiculous term to emphasize the silly nature of common arguments for making “open source” mean something it doesn’t mean to most people.

The openish source community claims that the movement towards providing free, source-code available programs will result in more secure software. This claim appears to be based on several fallacies briefly presented here: 1) the Microsoft fallacy, 2) the Java fallacy, and 3) the many-eyeballs fallacy.

The fact is that much of today’s software is badly behaved from many perspectives. It is insecure, unreliable, unsafe, buggy, poorly-designed, etcetera. Much work remains to be done to make software better. Whether something is openish source seems not to matter.

2. The Microsoft fallacy

The Microsoft fallacy is based on the following logic:

- 1) Microsoft makes bad software.
- 2) Microsoft software is closed source.
- 3) Therefore all closed source software is bad.

Without getting into the value judgment stated in premise 1, it is clear that this argument holds little water. Neither availability of source code nor cost and ownership of software have anything to do with software’s inherent goodness. Plenty of openish source software is bad. Plenty of closed source software is good.

Since Microsoft appears to be vilified by many software professionals, the Microsoft fallacy appeals to

raw emotions. Nevertheless it is not logical and should be abandoned as a supporting argument for openish source.

3. The Java fallacy

The Java fallacy can be stated as follows: If we keep fixing the holes in a give piece of software, eventually the software will be completely secure.

Openish source proponents interested in security implicitly assume that software is not a moving target. One lesson we can learn from the Java security field is that software is an evolving and dynamic thing. Since the target moves so quickly, assuming that patching holes will eventually result in something secure is incorrect.

To date, 19 serious security problems have been discovered in implementations of Java. Table 1 lists the holes and their accompanying exploits (by whimsical name). For information on each of the attacks, see [1]. Notice that a flurry of security problems historically accompanies a major release of Java (shown in bold face). Java 2, arguably one of the most secure commercially-viable platforms available, illustrates that a moving target does not become “more secure” over time.

Table 1: Java security holes resulting in attack applets.

JDK 1.0.7 (or so)

February 1996	Jumping the Firewall
March 1996	Slash and Burn
March 1996	Applets Running Wild
May 1996	Casting Caution to the Wind
June 1996	Tag-Team Applets
June 1996	You’re Not My Type
July 1996	Casting Caution to the Wind (reprise)
August 1996	Big Attacks Come in Small Packages

...

JDK 1.1

February 1997	Steal This IP Number
February 1997	Cache Cramming
March 1997	Virtual Voodoo
April 1997	The Magic Coat
May 1997	Verifying the Verifier
July 1997	The Vacuum Bug
August 1997	Look Over There

JAVA 2

July 1998	Beat the System
March 99	Verifier Hole
August 99	Race Condition
October 99	Verifier Hole II

4. The many eyeballs fallacy

My company, Reliable Software Technologies, recently released a source code based security scanner called ITS4 to the openish source community. Get a copy for yourself at <http://www.rstcorp.com/its4>.

Obviously, I believe that the code scanning approach we advocate in its4 integrates well with the openish source movement. From a security perspective, the main benefit of openish source software is what has been called the “many eyeballs” phenomenon, whereby a large number of developers pore over the code finding and repairing bugs in a more timely manner than would otherwise occur. This is a good thing. But note that releasing openish source software does not magically remove all the bugs from the program; nor is there any guarantee that any bugs at all will be found. These are the kinds of reasons that I believe static security analysis is not precluded by open source. Just for the record, I also believe that the use of a scanning tool like its4 does nothing to obviate the arguments in favor of open source.

Even some of the most widely scrutinized pieces of open source software have had significant bugs that lay undetected for years, despite numerous scrutinizing eyes. For example, an exploit and patch for a buffer overflow

condition in wu-ftpd was recently bandied about the net. This code has not only been scrutinized by numerous crackers, but it has also been used as a case study for a number of vulnerability detection techniques. FIST, a dynamic vulnerability detection tool based on fault injection, was able to identify the code as potentially exploitable over a year before the exploit was released [2]. However, before the exploit was publicized, the developers of FIST believed, based on their inspection of the code, that they had detected an un-exploitable condition.

There are other problems with relying on the outside world to debug security-critical code. When some security vulnerabilities are discovered, the problems remain hidden. Sometimes the person who first detects a vulnerability will keep the information private, and use it for nefarious purposes. Even when a vulnerability is patched or a new version of the once-broken software is released, there is still only a small chance that system administrators will apply the patch or install the new version.

5. Conclusion

Taken together, the Microsoft fallacy, the Java fallacy, and the many eyeballs fallacy provide a reasonable counter-argument to the usual openish source claims about security.

References

- [1] G. McGraw and E. Felten, *Securing Java: Getting down to business with mobile code*, John Wiley & Sons, NY, 1999.
A.B. Smith, C.D. Jones, and E.F. Roberts, “Article Title”, *Journal*, Publisher, Location, Date, pp. 1-10.
- [2] Anup Ghosh, Tom O’Connor, and Gary McGraw. An automated approach for identifying potential vulnerabilities in software. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 104-114, Oakland, CA, May 1998.