Application Security : The case for Open Source Software

Abstract

This paper examines the arguments made in the open source vs. closed source debate when it comes to security. It then contends that though merely being open-source does not make any software more secure, it definitely has an edge over closed source software.

1. Introduction

The task of securing internet services is an ongoing battle that definitely does not seem to be getting any easier over the years. As seen in Fig 1 below, the number of security incidents reported to CERT control center at Carnegie Mellon University, has been rising rapidly over the years.

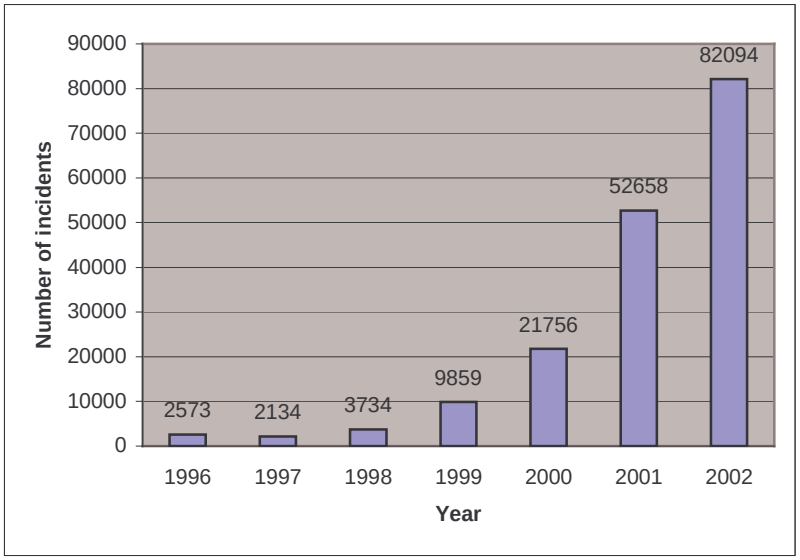

Fig1. Number of incidents reported per year to CERT. [cert]

Application software security is a very vulnerable link in the overall security of the enterprise. One of the questions to be asked when choosing software for your internet service is whether either of open source or closed source software has any advantage over the other when it comes to security.

According to Webopedia : "Generically, O*pen Source Software* (OSS) refers to a program in which the source code is available to the general public for use and/or modification from its original design free of charge, i.e., open. Open source code is typically created as a collaborative effort in which programmers improve upon the code and share the changes within the community.". *Closed Source Software* (CSS) on the other hand is a term used to refer to commercial-off-the-shelf (COTS) software for which no source code is available to the public; only the executable binaries.

The debate on the security of open vs. closed source software has been around for a while; with periodic arguments and counter arguments being made equally vocally and eloquently. The issue has been argued with empirical, anecdotal as well as conceptual arguments. The goal of this paper is ***not*** to show that all OSS is more secure than all CSS merely because it is open. Instead the goal is to show that choosing OSS when you're looking for secure software, can be, in many circumstances, a reasonable or even superior approach.

2. Countering the anti-OSS arguments

In this section, I will try to dispel some of the myths that have been built around the OSS concept and counter the arguments against OSS.

2.1 Someone could put a backdoor or trapdoor in open source software.

Critics argue that the adhoc nature of the development process implies that there is total disregard for tracking, accountability, or control and anybody could put a backdoor or trojan horse in the code.

Nothing could be farther from the truth because open-source developers use source control for coding. With large and diverse development teams being the norm in the open source world, source control is a *necessity,* not an option. Open source also uses code examination and analysis by others, and it puts the personal reputations of the authors on the line. Who would personally risk his or her reputation by putting a backdoor in source that is openly available in public forums?

This can be contrasted with closed source programs, which have an amazing array of "Easter eggs," those cute little surprises programmers leave in their code. What does the existence of such surprises say about the state of code review and source control in closed source circles? It does raise the question of trust. While we can always verify OSS source code, can we trust the binaries from CSS providers to not have such backdoors and trojans ? A concrete example is that of Borland's InterBase server [Cert-a]. 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". Without any external scrutiny of this code, Borland had no incentive to fix this flaw for nearly 6 years. Borland opened the source code for InterBase December 2000 and by January 2001 the advisory was up on CERT and the flaw was patched quickly.

2.2 Peer review in OSS is a myth and so flaws can abound

The OSS camp proclaims the source is available for anyone to examine and that volunteers do. The CSS camp counters that people either don't have the time or the skill or won't invest the effort to examine it. I will give an anecdotal evidence to illustrate how this generalization is not true. After the release of PGP 2.6, someone examining the code noticed an error in a random-number generator. The mistake was very minor. A statement that should have been an XOR with operation (~=) was in fact, an assignment (=).This didn't seriously compromise the security of PGP, but it did reduce the strength of the random keys. This error was quickly corrected, and the incident does illustrate some

important points. It shows that the code is examined by others and that coding errors (intentional or unintentional) do get spotted. Due to the minor, obscure, nature of this bug, it also indicates that the probability of a more serious bug or backdoor going undetected is rather low.

2.3 No one can be held liable for OSS

The argument is that there cannot be true liability for security or other flaws in OSS because there's no commercial entity to sue. But the most widely used commercial (closed-source) software today comes with an End User License Agreement (EULA) stipulating that the software's users accept that the vendor's only warranty is for the media used to distribute the software and that the vendor is not responsible for any problems caused by the software. So the liability argument doesn't hold water when made only against OSS.

2.4 Opening the source makes it easier for hackers to find security flaws

Well, this really isn't a myth. The real myth is that they won't find the security holes in closed source software because there is less information available to the attackers. Since this is the strongest anti-OSS argument I will counter it in detail.

A well-accepted premise in the cryptographic community is that there is no security through obscurity. As Kerckhoff noted about cipher systems in 1883 [Ker83], “the system must not require secrecy and can be stolen by the enemy without causing trouble." Kerckhoff's Principle doesn't speak of actual publication of the algorithms and protocols, just the requirement to make security independent of their secrecy.

In case of software we can show an even stronger premise that 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. Generally attackers (against both open and closed programs) start by knowing about the general kinds of security problems programs have. Attackers then use techniques to try to find those problems. Look at these techniques as ``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 like ``decompilers" that turn the machine code back into source code like output (resembling the original source code) and then search this source code for the vulnerable patterns (the same way they would search for vulnerabilities in open source software). So if an attacker wanted to use source code to find a vulnerability, a closed source program has no advantage.

3. Making the OSS case

3.1 Given enough eyeballs, all bugs are shallow

In his classic essay titled *"The Cathedral and the Bazaar"*, Eric Raymond laid out a law : "Given enough eyeballs, all bugs are shallow". He argues that given a large enough beta-tester and co-developer base, almost every problem in the software will be characterized quickly and the fix obvious to someone. This statement has been vehemently attacked by critics. They say that merely having many eyeballs look at code does not guarantee that they will find all flaws and that in fact a few expert eyes looking at the code are better than numerous random ones. Although we can agree that simply having several programmers looking at code will not ensure that they will study it carefully, in case of OSS there generally exists sufficiently large group of competent programmers who can be expected to care deeply: Those who either use the software personally or work for an enterprise that depends on it [Whit]. In fact, [Whit] goes on to say that auditing the programs on which an enterprise depends for its own security is a natural function of the enterprise's own information-security organisation.

3.2 Users can look for security flaws

In OSS there is an opportunity for interested parties to apply static analysis tools to detect the presence of bugs, malicious code or undocumented features. Automated tools can be applied to reduce the effort involved in looking for vulnerabilities. Since the source code is widely available, interested parties other than just the developer can perform "white box" testing. White-box test design allows one to peek inside the proverbial "box", and it focuses specifically on using internal knowledge of the software to guide the selection of test data.

3.3 Security vulnerability exposure tends to be shorter in OSS

Vulnerability exposure can be defined as the time between the announcement of the vulnerability and the time that a patch or other remedy was made available. It defines the minimum length of time that a software package is vulnerable to a particular method of attack. Assuming that the users of a software package install software updates for the package as soon as they are available, this metric measure the total length of time the users of the software packages are vulnerable to attack. The above assumption can be safely made because developers in both OSS and CSS do not control when users actually apply the patches.

Flaws and bugs (security related or otherwise) found in OSS can be quickly removed through the creation and distribution of software patches by the users themselves giving it a distinct edge. There is also empirical and anecdotal evidence that open source problems are often found and fixed by volunteers before they're widely exploited or at least faster than closed source flaws, while some closed source problem go unaddressed for months, or longer. For example, the "Ping 'O Death" bug was fixed in Linux only a few hours after it was announced. The same bug remained unsolved in some closed source systems for weeks or months. [Ritch] shows empirical evidence comparing Apache and IIS average vulnerability exposures over several years, declaring Apache as the winner.

3.4 Users can harden security

Individual user organizations can modify the source code to meet their own specific needs. For example, the NSA is developing tailored version of Linux by incorporating extra security features into the standard release [Nsa].

3.5 OSS model tends to improve code-quality

OSS programmers aren't generally paid to write the software they maintain. These programmers look for becoming well known by writing great software. They actually have stronger motivation to produce secure code than do programmers whose work will never be acknowledged because they work for large companies where their identity is unknown. Also, the OSS programmer's work -the source code- gets published, so the code written is subjected to peer review. Security-related bugs or inclusion of backdoors can seriously hurt his/her credibility.

3.6 Empirical evidence

There are a lot of statistics available [Attr-a] [Attr-b] [Sec-Foc] [Cert] [Mi2g-a] [Ritch] to illustrate the relative disposition of security vulnerabilities across OSS and CSS softwares/Operating systems. I prefer to focus on conceptual arguments and only provide the following two supporting charts because statistics can often be spun either way and also merely having a larger number of vulnerabilities does not mean a particular system is less secure because the severity of these flaws should also be taken into account.

Fig2. shows that the total number of (successful) digital attacks on Linux systems are generally lower than those on Windows based systems [Mi2g-a]

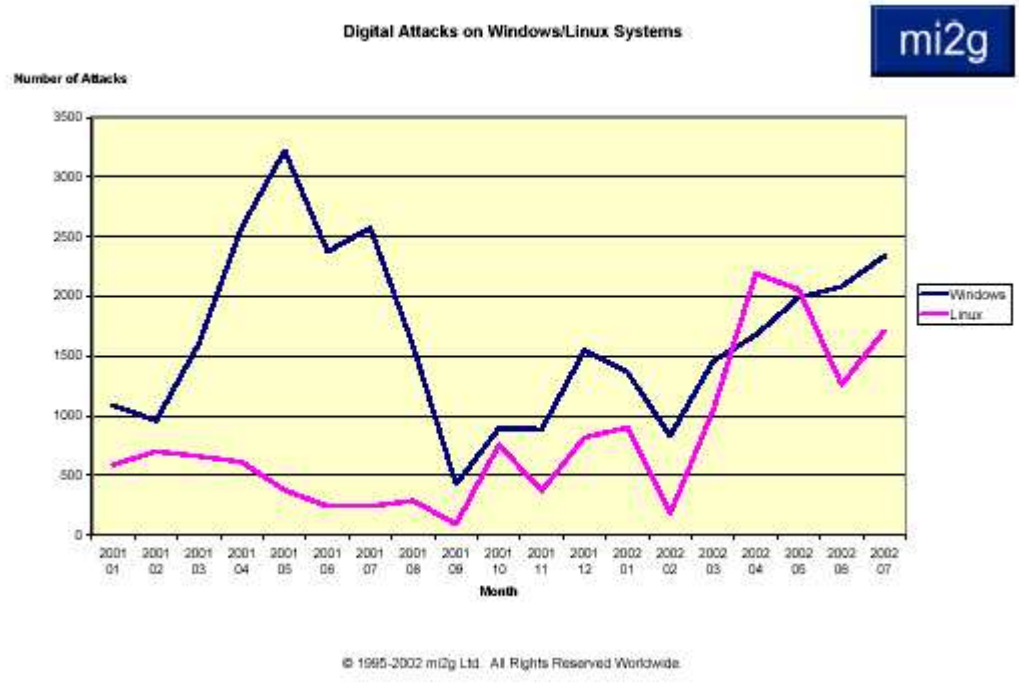

Fig2. Comparison of attacks on Linux vs. Windows in 2001-2002 [Mi2g-a]

Fig3. shows a chart based on data collected in [Ritch] between 1996-2001 that shows on an average Apache having less severe vulnerabilities as compared to the closed source IIS webserver.

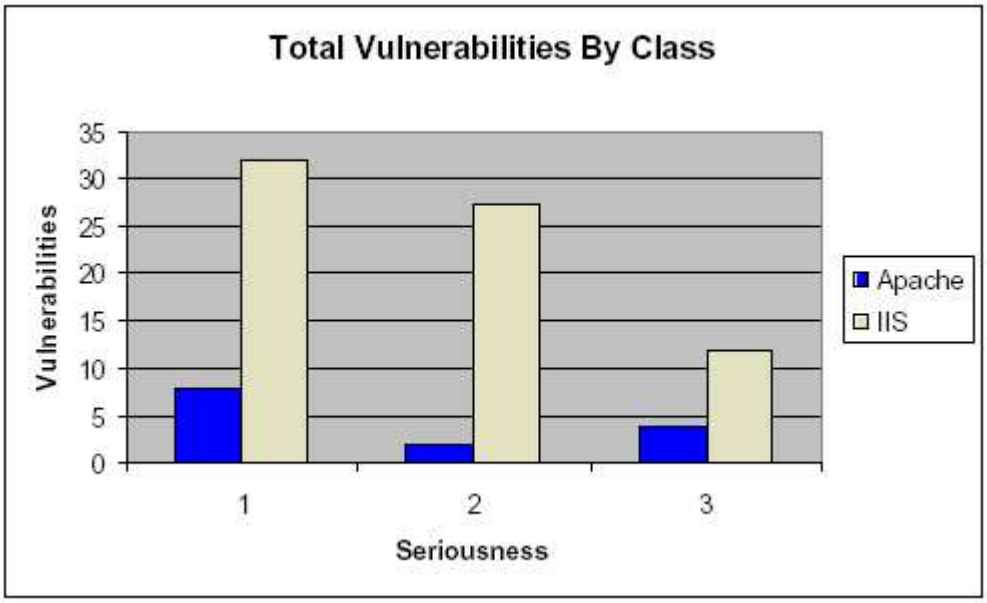

Fig3. Apache vs. IIS vulnerabilities (1 is most serious)

4. Conclusion

Security and trust go hand-in-hand and it is obvious that trusting closed binaries to be secure is harder than trusting open source code simply because the open code is open to scrutiny. One should be careful in trusting OSS code as well; just because the source is open does not mean one can trust the corresponding binary. For example, Ken Thomson, during his 1983 Turing Award lecture to the ACM, revealed a shocking, and subtle, software subversion technique [Thom]. Thompson had modified the UNIX C compiler to recognize when the login program was being compiled, and to insert a back door in the resulting binary code such that it would allow him to login as any user using a "magic" password. Anyone examining the login program code would not have caught this backdoor because the code only existed in the binary. Anyone reviewing the compiler source code could have found the back door, except that Thompson then modified the compiler so that whenever it compiled itself, it would insert both the code that inserts the login back door, as well as code that modifies the compiler. With this new binary he removed the modifications he had made and recompiled again. He now had a trojaned compiler and clean source code. Anyone using his compiler to compile either the login program , or the compiler, would propagate his back doors.

To conclude, open source software is certainly not a security panacea, but it is definitely a better choice when pitted against closed source software.

References

[Ray] Eric Raymond. *The Cathedral and the Bazaar*. http://www.commercialos.org/article.phtml/14

[War98] Michael H. Warfield. *Musings on open source security model*. http://www.linuxworld.com/linuxworld/lw-1998-11/lw-11-ramparts_p.html

[And02] R. Anderson (2002). *Security in Open versus Closed Systems - The Dance of Boltzmann, Coase and Moore*. http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/toulouse.pdf

[Whee03] David A. Wheeler. *Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!*.http://www.dwheeler.com/oss_fs_why.html

[Ritch] Ronald W Ritchey. *Open Source Vs. Close Source Software An Experiment To Determine Which is More Secure.* http://www.isse.gmu.edu/faculty/ofut/classes/763/studpapers/Ritchey-1106.pdf

[Mi2g-a] Mi2g report (2002). *Windows regains mantle of most vulnerable OS.* http://www.mi2g.com/cgi/mi2g/frameset.php?pageid=http%3A//www.mi2g.com/cgi/mi2g/press/140802.php

[Mi2g-b] Mi2g report (2002) *Digital attacks on Open Source systems soar* http://www.mi2g.com/cgi/mi2g/frameset.php?pageid=http%3A//www.mi2g.com/cgi/mi2g/press/110702.php

[Thom] Ken Thomson. *Reflections on Trusting Trust.* http://www.acm.org/classics/sep95/

[Whit] Diffie Whitfield. *Risky Business : Keeping security a secret.* http://www.zdnet.com.au/builder/program/unix/story/0,2000034968,20271338,00.htm

[Ker83] Kerckhoff, A.(1883) *La Cryptographie Militaire*. Journal des Sciences Militaires http://www.cl.cam.ac.uk/users/fapp2/kerckhoffs/

[Attr-a] http://www.attrition.org/mirror/attrition/os-graphs.html

[Attr-b] http://www.attrition.org/mirror/attrition/os.html

[Sec-foc] http://www.securityfocus.com/vulns/stats.shtml

[Cert] http://www.cert.org/stats/

[Cert-a] http://www.cert.org/advisories/CA-2001-01.html

[Nsa] http://www.nsa.gov/selinux/