INCREASED SECURITY Through Open Source

It may seem counterintuitive, but going “open” all the way offers the most security.

The last few years have shown a worldwide rise in attention toward, and actual use of, open source software (OSS), most notably the operating system Linux and various applications running on top of it. Various major companies and governments are adopting OSS, and as a result, there are many publications concerning its advantages and disadvantages. The ongoing discussions cover a wide range of topics, such as Windows versus Linux, cost issues, intellectual property rights, and development methods. Here, we focus on security issues surrounding OSS. It has become a reasonably well-established conviction within the computer security community that publishing designs and protocols contribute to the security of systems built on them. But should one go all the way and publish source code as well? That is the fundamental question that we address in this article.

The following analogies may help to introduce the issues and controversies surrounding the open source debate:

• While traveling far from home, would you take medicines of an unknown brand given to you by a self-proclaimed “doctor,” without documentation, and hence without (independent) assurance about the nature and mechanisms of action of the medication?

• Who would you trust most? A locksmith who keeps the workings of his locks secret so thieves cannot exploit this knowledge? Or a locksmith who publishes the workings of his locks so everyone (including thieves) can judge how good/bad they are (relying exclusively on the complexity of the keys for protection)?

In this article we discuss the impact of open source on both the security and transparency of a software system. We focus on the more technical aspects of this
issue (and refer to Glass [4] for a discussion of the economical perspective of open source), combining and extending arguments developed over the years [12]. We stress that our discussion of the problem only applies to software for general-purpose computing systems. For embedded systems, where the software usually cannot easily be patched or upgraded, different considerations may apply.

**Security Through Obscurity:**

**Design vs. Implementation**

Through the centuries, secrecy was the predominant methodology surrounding the design of any secure system. Security of military communication systems, for example, was mostly based on the fact that only a few people knew how it worked, and not on any inherently secure method of communication. Ciphers in those days were not particularly difficult and, in fact, for the purpose of this discussion we need to distinguish between the security of a system, the exposure of that system, and the risk associated with using that system.

The ultimate decisive factor that determines whether a system is “secure enough” is the risk associated with using that system. This risk is defined as a combination of the likelihood of a successful attack on a system together with the damage to assets resulting from it. The exposure of a system completely ignores the damage that is incurred by a successful attack, and is defined simply as the likelihood of a successful attack. This depends on several factors, like the number and severity of vulnerabilities in the system, but also whether these vulnerabilities are known to attackers, how difficult it is to exploit a vulnerability, and whether or not the system is a high-profile target.

Finally, we consider the security of a system to be an objective measure of the number of its vulnerabilities and their severity (that is, the privileges obtained by exploiting the vulnerability).

To summarize, exposure combines security with the likelihood of attack, and risk combines exposure with the damage sustained by the attack. We note that in other papers on this and similar topics, security has been used to mean either security proper, or exposure, or risk as defined previously. With these definitions in place, we see that opening the source clearly does not change the security of a system (simply because it doesn’t introduce new bugs), while the exposure is likely to increase in the short term (because it makes the existing bugs more visible).

**WHo WOULD YOu TRuST MoSt?**

A locksmith who keeps the working of his locks secret so thieves cannot exploit this knowledge? Or a locksmith who publishes the workings of his locks so everyone (including thieves) can judge how good/bad they are? There is no agreement on the answer to this question even in the academic community [1]. From our perspective, the answer is: “absolutely!” Here we will argue why.

**Security, Risk, and Exposure**

When discussing whether open source makes systems more secure, we have to be precise about what we mean. In fact, for the purpose of this discussion we need to distinguish between the security of a system, the exposure of that system, and the risk associated with using that system.

The ultimate decisive factor that determines whether a system is “secure enough” is the risk associated with using that system. This risk is defined as a combination of the likelihood of a successful attack on a system together with the damage to assets resulting from it. The exposure of a system completely ignores the damage that is incurred by a successful attack, and is defined simply as the likelihood of a successful attack. This depends on several factors, like the number and severity of vulnerabilities in the system, but also whether these vulnerabilities are known to attackers, how difficult it is to exploit a vulnerability, and whether or not the system is a high-profile target.

Finally, we consider the security of a system to be an objective measure of the number of its vulnerabilities and their severity (that is, the privileges obtained by exploiting the vulnerability).

To summarize, exposure combines security with the likelihood of attack, and risk combines exposure with the damage sustained by the attack. We note that in other papers on this and similar topics, security has been used to mean either security proper, or exposure, or risk as defined previously. With these definitions in place, we see that opening the source clearly does not change the security of a system (simply because it doesn’t introduce new bugs), while the exposure is likely to increase in the short term (because it makes the existing bugs more visible).

The question is what happens to the security and the exposure of an open source system in the long run.

**Open vs. Closed Source**

The increased attention paid to open source in the media and by society at large has made open source an almost catch-all phrase. Here, we use it in its original, rather specific, meaning. Open source software is software for which the corresponding source (and all relevant documentation) is available for inspection, use, modification, and redistribution by the users. We do not necessarily distinguish between any kind of development methodology (for example, the Cathedral or the Bazaar [10]). Nor do we care about the pricing model (freeware, shareware, among others). We do assume, however, that users (in principle) are allowed and able to rebuild the system from the (modified) sources, and that they have access to the proper tools to do so.

In some cases, allowing the user to redistribute the modified sources (in full or through patches) is also necessary (for example, Free Software and the GNU Public License*). Most of our arguments also hold for source available software, where the license does not allow redistribution of the (modified) source.

**Open Source for Security**

We argue that open source software is a necessary requirement to build systems that are more secure. Our main argument is that opening the source allows independent assessment of the exposure of a system and the risk associated with using the design. As government and closed-source corporations realign themselves looking error-checking feature added to a system call.

Finally, and more generally, the quality of a piece of software (and patches to it) depends on the skills of the programmers working on it [8]. For many open source projects there is no a priori selection of programmers based on their skill. Usually any help is appreciated, and there is only rudimentary quality control.

**Arguments against closed source.**

Let us first review the arguments put forward against open source in the previous section. The last two arguments against open source are actually aimed at the development methodology instead. The systems developed in that manner need to patch all vulnerabilities to protect himself from the attacker. The attacker must find but one vulnerability to successfully attack the system. The defender needs to patch all vulnerabilities to protect himself completely. This is considered an uneven battle.

Fourth, there is no direct guarantee that the binary object code running in the computer corresponds to the source code that has been evaluated [11]. People unable or unwilling to compile from source must rely on a trusted third party to vouch for this.

Also, making the source public does not guarantee that any qualified person will actually look at the source and evaluate (let alone improve) it. There are many open source projects that, after a brief flurry of activity, are only marginally maintained and quickly sink into oblivion. The attackers, on the other hand, most surely will scrutinize the source.
We do not distinguish between any kind of security. Let us first review January 2007/Vol. 50, No. 1 81.”

**Security, Risk, and Exposure**
When discussing whether open source makes systems more secure, we have to be precise about what we mean. In fact, for the purpose of this discussion we need to distinguish between the security of a system, the exposure of that system, and the risk associated with using that system.

The ultimate decisive factor that determines whether a system is "secure enough" is the risk associated with using that system. This risk is defined as a combination of the likelihood of a successful attack on a system together with the damage to assets resulting from it.

The exposure of a system completely ignores the damage that is incurred by a successful attack, and is defined simply as the likelihood of a successful attack. This depends on several factors, like the number and severity of vulnerabilities in the system, but also whether these vulnerabilities are known to attackers, how difficult it is to exploit a vulnerability, and whether or not the system is a high-profile target.

Finally, we consider the security of a system to be an objective measure of the number of its vulnerabilities and their severity (that is, the privileges obtained by exploiting the vulnerability).

To summarize, exposure combines security with the likelihood of attack, and risk combines exposure with the damage sustained by the attack. We note that in other papers on this and similar topics, security has been used to mean either security proper, or exposure, or risk as defined previously. With these definitions in place, we see that opening the source clearly does not change the security of a system (simply because it doesn’t introduce new bugs), while the exposure is likely to increase in the short term (because it makes the existing bugs more visible).

**Open Source for Security**
We think that opening source software is a necessary requirement to build systems that are more secure. Our main argument is that opening the source allows independent assessment of the exposure of a system and the risk associated with using the system, and that this is necessary to successfully design and implement systems.

In our case in detail.

**WHO WOULD YOU TRUST MOST?**
A locksmith who keeps the working of his locks secret so thieves cannot exploit this knowledge? Or a locksmith who publishes the workings of his locks so everyone (including thieves) can judge how good/bad they are?

The question is what happens to the security and the exposure of an open system in the long run.

**Open vs. Closed Source**
The increased attention paid to open source in the media and by society at large has made open source an almost catch-all phrase. Here, we use it in its original, rather specific, meaning. Open source software is software for which the corresponding source (and all relevant documentation) is available for inspection, use, modification, and redistribution by the users. With respect to security, there is no clear distinction between any kind of development methodology (for example, the Cathedral or the Bazaar [10]). Nor do we care about the pricing model (freeware, shareware, among others). We do assume, however, that users (in principle) are allowed and able to rebuild the system from the (modified) sources, and that they have access to the proper tools to do so.

In some cases, allowing the user to redistribute the modified sources (in full or through patches) is also necessary (for example, Free Software and the GNU Public License). Most of our arguments also hold for source available software, where the license does not allow redistribution of the (modified) source.

Arguments against closed source.

Finally, and more generally, the quality of a piece of software (and patches to it) depends on the skills of the programmers working on it [8]. For many open source projects there is no a priori selection of programmers based on their skill. Usually any help is welcomed, and then the most likely, and forces software developers to spend more effort on the quality of their code. Here, we argue our case in detail.

We will first review arguments in favor of keeping the source closed, and then discuss reasons why open source does (in the long run) increase security. As noted earlier, there is a distinction between making the design of a system public and also making its implementation public. We focus on the latter case, but note that most (but not all) of these arguments also apply to the question of whether or not the design should be open.

Keeping the source closed. First of all, keeping the source closed prevents the attacker from having easy access to information that may be helpful to successfully launch an attack [2]. Opening the source gives the attacker a wealth of information to search for vulnerabilities and/or bugs, like potential buffer overflows, and thus increases the exposure of the system.

Also, there is a huge difference between openness of the design and openness of the source. The openness of the design may reveal logical errors in the security in the worst case. With proper review, these errors can and usually are found. For source code, this is not, or at least not completely, the case. In the foreseeable future, source code will continue to contain bugs, no matter how hard we look, test, or verify.

Moreover, opening the source gives unfair advantage to the attacker. The attacker must find but one vulnerability to successfully break into the system. The defender needs to patch all vulnerabilities to protect himself completely. This is considered an uneven battle.

Fourth, there is no direct guarantee that the binary object code running in the computer corresponds to the source code that has been evaluated [11]. People unable or unwilling to compile from source must rely on a trusted third party to vouch for this.

Also, making the source public does not guarantee that any qualified person will actually look at the source and evaluate (let alone improve) it. There are many open source projects that, after a brief flurry of activity, are only marginally maintained and quickly sink into oblivion. The attackers, on the other hand, most surely will scrutinize the source.

In a closed style, back doors may be introduced into the source by hackers posing as trustworthy contributors. That this is not an idle threat became clear in November 2003, when Linux kernel developers discovered a back door in a harmlessly looking error-checking feature added to a system call.

Finally, and more generally, the quality of a piece of software (and patches to it) depends on the skills of the programmers working on it [8]. For many open source projects there is no a priori selection of programmers based on their skill. Usually any help is appreciated, and there is only rudimentary quality control.

Arguments against closed source. Let us first review the arguments put forward against open source in the previous section. The last two arguments against open source are actually aimed at the development methodology instead. The strength of the case against open source would also be more insecure if they were closed source. We assume a minimal standard of proper coding practices, project management, change control and quality control. In fact, one of our main points is

Or could one work like systems to confine confidential object code, and to enforce a security policy irrespective [9].

See [www.opensource.org]

See [www.gnu.org/software/gpl.html]
that by opening up the source, software projects cannot get away with poor project management and poor quality control so easily.

Turning to the first argument against closed source, we note that keeping the source closed for a long time appears to be difficult [7]. Last year, the source code for certain types of voting machines manufactured by Diebold were distributed on the Internet, and subsequent research on that source code revealed horrible programming errors and security vulnerabilities [6]. Recently, even parts of the source to Microsoft Windows NT became public. Within days the first exploit based on this source code was published. The Diebold case also revealed how inferior the coding standards of current closed source systems can be, and how they lead to awfully insecure systems.

Even if the source remains closed, vulnerabilities of such closed source systems will eventually be found and become known to a larger public after a while. Vulnerabilities in existing closed source software are announced on a daily basis. In fact, tools like debuggers and disassemblers allow attackers to find vulnerabilities in applications without access to the source relatively quickly. Moreover, not all vulnerabilities that are discovered will be published. Their discoverers may keep them secret to avoid patches for them, allowing use of the vulnerability to exploit systems for a prolonged period of time. We see that while the perceived exposure of a closed source system may appear to be low, the actual exposure eventually becomes much higher (approaching the exposure of open source systems if it was open initially if the system was completely open source).

Even worse, the only producer of closed source software can release patches for any vulnerabilities that are found. Many of those patches are released weeks or months after the vulnerability is discovered, if at all. The latter case occurs, for instance, with legacy software for which the company producing it either no longer exists or refuses to support it after a while (as with Microsoft Windows NT Server 4.0 and Netscape Calendar, for example). The consequence is that systems stay exposed longer, increasing the risk of using that system.

We see that keeping the source closed actually hurts the defender much more than the attacker: while a determined attacker can still discover weaknesses easily, the defender is prevented from patching them.

Finally, closed source software severely limits the ability of the user of such software to evaluate its security for or by himself. The situation improves if at least the design of the system is open. If the system is evaluated by an independent party according to some generally accepted methodology (like the Common Criteria), this gives the user another basis for trusting the security of the software. However, such evaluations are rare (because they are expensive), and usually limited to certain restricted usage scenarios or parameter settings that may not correspond to the actual operating environment of a particular user. Moreover, such evaluations apply only to a specific version of the software: new versions need to be reevaluated.

The way forward: Arguments supporting open source.

We see that the arguments against "security through obscurity" generally apply to the implementation of a system as well. A widely held design principle that the security of a system should only depend on the secrecy of the (user-specific) keys on the grounds that all other information of the system is shared by many other people and therefore will become public as a matter of course. Moreover, if the source is open users can use the software by themselves, or to hire a party of their choice to evaluate the software for them. Open source even enables several different and independent teams of people to evaluate the security of the system.

Open source enables users to evaluate the security by themselves, or to hire a party of their choice to evaluate the security for them. Open source even enables several different and independent teams of people to evaluate the security of the system.

Open source enables users to find bugs, and to patch bugs themselves. There is also a potential network effect: if users submit their patches to a central repository, all other users can update their system to include this patch, increasing their security too. Given that different users are likely to find different bugs, many bugs are potentially removed. This leads to more and faster patches, and hence more secure code (this corresponds to "Linus’s Law": “Given enough eyeballs, bugs are shallow” [10]). Evidence suggests that patches for open source software are released almost twice as fast as for closed source software, thus halving the vulnerability period [12].

If a user is unable to patch a bug himself, open source at least enables him to communicate about bugs with developers more efficiently (because both can use the same frame of reference—that is, the code for—communication [10]).

Also, open source software enables users to add extra security measures. Several tools exist to enhance the security of existing systems, provided the source is available [3]. These tools do not rely on static checking of the code. Instead, they add generic runtime checks to the code to detect, for example, buffer overflows or stack frame corruptions. Moreover, developers of open source software allow all the user to limit the complexity of the system (thereby increasing its security) by removing unneeded parts.

Finally, and importantly, open source forces developers communities to be more careful, and to use the best possible tools to secure their systems. It also forces them to use clean coding styles (“sloppy code is untrustworthy”), and to put more effort into quality control. Otherwise, companies and individual programmers alike will lose respect and creditibility. As a side effect, this will stimulate research and development in new, improved tools for software development, testing, and evaluation, and perhaps even verification.

Conclusion

We conclude that opening the source of existing systems will at first increase their exposure, due to the fact that information about vulnerabilities becomes available to attackers. However, this exposure (and the associated risk of using the system) can now be determined publicly. With closed source systems the exposed actual exposure may appear to be low, while the actual exposure (due to increased knowledge of the attackers) may be much higher. Moreover, because the source is open, all interested parties can assess the exposure of a system, thereby increasing its security. New, much more advanced, tools will be developed to improve the security of such systems. Even further, open source allows users to make a more informed choice about the security of a system, based on their own or on independent judgment.

It is our conviction that all these benefits outweigh the disadvantages of a short period of increased exposure.

References

1. Anderson, R. Security in open versus closed systems—The dance of Boltzmann, Caesar and Moses. In Proceedings of 9th Conference on Open Source Software Economics (Toulouse, France, June 20–21 2002).
2. Brown, K. Opening the open source delusion. Technical report, Abhin of Tricorpundat Institute, June 2002.
3. Cowan, C. Software security for open-source systems. IEEE J. Secur. & Privacy 1, 1 (2003), 38–45.
4. Glass, R.L. A look at economics of open source. Comm. ACM 47, 2 (Feb. 2004), 25–27.
5. Kerckhoff, A. La cryptographie militaire. Journal de sciences militaires, (June 1883), 48–49.
6. Kohno, T., Stubblefield, A., Rubin, A.D., and Wallach, D.S. Analysis of an electronic voting system. In Proceedings of the IEEE Conference on Security & Privacy. (Oakland, CA, May 9–12 2004).
7. Menzies, R.T., and Neumann, P.G. Inside Risks: Security by obscurity. Comm. ACM 46, 11 (Dec. 2003), 160.
8. Neumann, P.G. Inside Risks: Information system security roles. ACM Trans. Inf. Syst. Security 3, 1 (Oct. 2000); 1:36.
9. Peres, N. Improving host security with system call policies. In Pre- ceedings of the 12th USENIX Security Symposium (Washington D.C., Aug. 2003).
10. Raymond, E.S. The Cathedral and the Bazaar. 2000.
11. Thompson, K. Reflections on trusting trust. Comm. ACM 27, 8 (Aug. 1984), 761–765.
12. Wiman, B., Lundberhe, C., and Calyamides, M. Does open source system improve system security? IEEE Software, (Sept. –Oct. 2001), 51–67.
that by opening up the source, software projects cannot get away with poor project management and poor quality control so easily.

Turning to the first argument against closed source, we note that keeping the source closed for a long time appears to be difficult [7]. Last year, source code for certain types of voting machines manufactured by Diebold were distributed on the Internet, and subsequent research on that source code revealed horrible programming errors and security vulnerabilities [6]. Recently, even parts of the source to Microsoft Windows NT disappeared resulting in either no longer exists or refuses to support it after a while (as with Microsoft Windows NT Server 4.0 and Netscape Calendar, for example). The consequence is that systems stay exposed longer, increasing the risk of using that system.

We see that keeping the source closed actually hurts the defender much more than the attacker: while a determined attacker can still discover weaknesses easily, the defender is prevented from patching them.

Finally, closed source software severely limits the ability of the user of such software to evaluate its security for or by himself. The situation improves if at least the design of the system is open. If the system is evaluated by an independent party according to some generally accepted methodology (like the Common Criteria), this gives the user another basis for trusting the security of the software. However, such evaluations are rare (because they are expensive), and usually limited to certain restricted usage scenarios or parameter settings that may not correspond to the actual operating environment of a particular user. Moreover, such evaluations apply only to a specific version of the software: new versions may be reevaluated.

The way forward: Arguments supporting open source.

We see that the arguments against "security through obscurity" generally apply to the implementation of a system as well. It is a widely held design principle that the security of a system should only depend on the secrecy of the (user-specific) keys, on the grounds that all other information of the system is shared by many other people and therefore will become public as a matter of course.

Moreover, open source enables users to evaluate the system by themselves, or to hire a party of their choice to evaluate the system for them. Open source even enables several different and independent teams of people to evaluate the security of the system.

Conclusion

We conclude that opening the source of existing systems will at first increase their exposure, due to the fact that less information about vulnerabilities becomes available to attackers. However, this exposure (and the associated risk of using the system) can now be determined publicly. With closed source systems, the received exposure may appear to be low, while the actual exposure (due to increased knowledge of the attackers) may be much higher.

Moreover, because the source is open, all interested parties can assess the exposure of a system, which can use the same frame of reference—that is, the source code—for communication [10].

Open source software enables users to find bugs, and to patch bugs themselves. There is also a potential network effect: if users submit their patches to a central repository, all other users can update their system to include this patch, increasing their security too. Given that different users are likely to find different bugs, many bugs are potentially removed. This leads to more rapid and faster patches, and hence more secure code (this corresponds to “Linus’s Law”: “Given enough eyeballs, bugs are shallow” [10]). Evidence suggests that patches for open source software are released almost twice as fast for as closed source software, thus halving the vulnerability period [12].

If a user is unable to patch a bug himself, open source at least enables him to communicate about bugs with developers more efficiently (because both can use the same frame of reference—that is, the source code—for communication [10]).

Also, open source software enables users to add extra security measures. Several tools exist to enhance the security of existing systems, provided the source is available [3]. These tools do not rely on static checking of the code. Instead, they add generic runtime checks to the code to detect, for example, buffer overflows or stack frame corruptions. Moreover, the source of the software allows the user to limit the complexity of the system (thereby increasing its security) by removing unneeded parts.

Finally, and importantly, open source forces developer communities to be more careful, and to use the best possible tools to secure their systems. It also forces them to use clean coding styles (“sloppy” code is untrustworthy), and to put more effort into security issues. Open source is thus a natural fit for improving software and security development, and perhaps even verification.

Open source enables users to evaluate the system by themselves, or to hire a party of their choice to evaluate the security for them. Open source even enables several different and independent teams of people to evaluate the security of the system.

References

1. Anderson, R. Security in open versus closed systems—The dance of the unforeseen. In Proceedings of the 1st Conference on Open Source Software Economics (Toulouse, France, June 20–21 2002).
2. Brown, K. Opening the open source debate. Technical report, Alexis de Tocqueville Institute, June 2002.
3. Cowan, C. Software security for open-source systems. IEEE J. Secur. Privy 1 (2003), 38–100.
4. Glass, R.L. A look at the economics of open source. Comm. ACM 47, 2 (Feb. 2004), 25–27.
5. Kerckhoff, A. La cryptographie militaire. Journal de sciences militaires, 7 (1883), 161–191 (Also Jan. 1883, 161–191 and Feb. 1883).
6. Kohno, T., Stallings, A., Rubin, A.D., and Wallach, D.S. Analysis of an electronic voting system. In Proceedings of the IEEE Conference on Security & Privacy (Oakland, CA, May 9–12 2004).
7. Messiah, R.T., and Naumann, P.G. Inside Risk: Security by obscurity. Comm. ACM 46, 11 (Dec. 2003), 160.
8. Naumann, P.G. Inside Risk: Information system security roles. Comm. ACM 44, 10 (Oct. 2001), 186.
9. Provos, N. Improving host security with system call policies. In Proceedings of the 2nd USENIX Symposium (Washington D.C., Aug. 2003).
10. Raymond, E.S. The Cathedral and the Bazaar. 2000.
11. Thompson, K. Reflections on trusting trust. Comm. ACM 27, 8 (Aug. 1984), 760–768.
12. Wiman, B., Landhede, C., and Caleyand, M. Does open source improve system security? IEEE Secure., Sept.–Oct. 2001, 57–61.

JAN-HENK HOFMANN (shofmann.nl) is a senior researcher in the Security of Systems Group at the Institute for Computing and Information Sciences of the Radboud University, Nijmegen, The Netherlands.

BART JACOBS (bartv.nu) is a Professor of Software Security and Correctness and Research Director of the Institute for Computing and Information Sciences of the Radboud University, Nijmegen, The Netherlands.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

© 2007 ACM 0002-0727/07/0100-0390 $5.00