Is Open Source Software More Secure?

Russell Clarke
russellc@microsoft.com

David Dorwin
ddorwin@u.washington.edu

Rob Nash
rundaemon@gmail.com

Homeland Security / Cyber Security

Contents

Abstract

In an attempt to contribute to the current state of understanding with respect to systems security, this paper inspects one software source distribution philosophy that underlies the operation of a representative class of networked computers today. Establishing whether open source leads to more secure software will have serious implications for organizations utilizing or constructing open source software, the trust established between a user and a program (irrespective of source visibility), and provide valuable observations for proprietary software vendors as well. It is the intent of this paper to advance forward the state of understanding with respect to source philosophies, initially and critically explore the vulnerability differences caused by the source visibility differences, and propose an answer to a modern and relevant question: Does open source development lead to more secure code?

1 Introduction

In this paper, we seek to measure the security value (or lack thereof) of the open source software distribution philosophy. While many groups treat this discussion as a religious debate between open source and proprietary software, we seek to empirically describe the issues and factors in support of or against the security of open source software and avoid as best we can the issues we cannot measure. Further, this debate is often partitioned along the axis of the dominant operating system on each side, and may be colored by opinions just as much as fact. However, we believe the security analysis of, and differences between, the open source and proprietary software design philosophies dont reduce to a contest between Microsoft Windows (proprietary) and Apple Macintosh OS X (where the kernel is primarily open source) or Linux (completely open source). Rather, in this paper we seek to observe the theoretical and practical differences in security between representative classes of software systems, regardless of the corporations or organizations responsible for the products under scrutiny. Hence, the examples presented in this paper are chosen as practical, representative, and timely. We feel it is appropriate, then, to recognize and understand the tussles that exist in this area of research, to then formulate our reasoning and research further. Also, since there is both fiction and fact on each side of the debate, we initially advocate for both sides, only to later analyze and explain these observations.

1.1 Scope

This section defines the extent and limits of our discussion. We have chosen many issues as topics for future work, and identified issues not bearing significant relevance for their inclusion, understanding that time, trials, and experience will validate or discredit these simplifying assumptions. We are not interested in

the dissection of the corporate ecosystems, nor in comparisons of one company's product versus another (indeed, such reports are already plentiful). We are not arguing which operating system is superior overall, which has more features, or which is more economical. We do not even argue that one system is more secure than another, but rather, we seek out data in an attempt to qualitatively (more data here) and quantitatively (less data here) confirm or deny the security of open source software due to its inherent code visibility. Topics peripheral to our study of cyber security, such as licensing for open source and closed source systems, have been ommitted. Finally, we recognize that significant changes to the state of the art, such as stronger tools, breakthroughs in quantum computing, types of attacks not previously known, underlying network protocol alterations (via deployment of Internet2 or IPv6, for example) will have significant ramifications to what is researched and presented here.

2 Initial Systems Analysis

2.1 A Brief Overview of Source Theologies

In this section we present the two opposing views on the security of open source. The intent is to summarize the view of each side of the argument, regardless of whether it can be proven. In later sections we address these claims with facts and data.

2.1.1 Open Source is More Secure

With open source, any who wish to see the source code for any part of project can do so. Bugs including security vulnerabilities may be spotted by the many eyes – both the experts and novices alike – on the code. Open source code is subject to security reviews (as in any professional software developing enterprise), but

in addition to "in-house" reviews by those engineers tied to the project, is also subject to unsolicited security reviews that may be conducted by anyone in the world. One could argue that, in general, more code walkthroughs are likely to occur in open source projects, since usually an entity responsible for a project will test it to some amount and outside sources will also test your source some amount, potentially resulting in more code scrutiny in this paradigm. This is especially true when large corporations have a vested interest in or sell products based on an open source project. In this case, they have the same interest in the project being secure as if it were proprietary. Finally, while security is oft cited as a primary reason for not opening source code to the public, there seems to be little conclusive evidence to support this.

2.1.2 Security Through Obscurity

This argument is as follows: If I hide my code from my adversaries, it will be more difficult for them to identify vulnerabilities in my software. Only negative results can come from offering visibility at the source level. Writing good software is difficult, and writing nontrivial, flawless software is probably impossible. Hence, if software is complex to generate, and at least somewhat complex to reverse engineer, why sacrifice protection afforded to us by obscurity? Certainly, the technique is not mutually beneficial; it simply provides the blueprints of your construction to your adversary. As tools for identifying security flaws improve, these will also benefit the adversary as they can identify vulnerabilities in code bodies before fixes can be deployed to customers.

If I could slow or prevent the propagation of my source, then my organization can capitalize on the gains that we are exclusively responsible for.

As with any battle, giving your plans to your adversaries or competitors is not a good strategy for security or business in general.

2.2 Metrics are Often Misleading

Deciding this debate would be much simpler if we could just look at the number of attacks against or reported vulnerabilities in comparable open source and proprietary software products and declare a winner. However, there are many variables besides development philosophy that affect such metrics. The size of the deployed base, code size, features provided, business decisions, effectiveness of customer maintenance and patching, and adversaries' preferences all affect these statistics. It is in an adversary's interest to exploit vulnerabilities present in as many systems as possible. Thus, she may choose to identify and exploit security vulnerabilities in the most popular operating systems or applications. Likewise, it is unlikely that security organizations and "white hats" put equal amounts of effort into identifying vulnerabilities in all software (even the leading products). As such, we are unable to conclude directly that a software product with more reported attacks or vulnerabilities is, in fact, more or less secure than another product from this information alone.

2.3 Open and Closed Source Software are Intertwined

In one example[1], a vulnerability was identified, reported, and fixed in the Linux operating system. Upon release of the security update, attackers learned the exact nature of the vulnerability and conjectured a similar vulnerability might exist in other resource managers and operating systems. That group was successful in constructing an attack exploiting the same vulnerability that was indeed present in the closed source operating system, Windows NT. So, we see that the existence of an open source program (and its constituent security updates) can reveal the vulnerabilities of both itself and any isometric system. This is implicit, obvious, and important in that it could suggest, for example, that open source software and closed source software are intertwined

in a way that makes investigation difficult. While it is the case that these systems do not operate in isolation of one another, and have each been constructed with many engineering standards and similarities, we conjecture that it is not impossible to separate the two systems for analysis, and further, that this evidence might suggest that we can leverage the similarity of systems in our study by observing wide classes of vulnerabilities in a platform-independent fashion. Further, we should consider the effects of releasing patches out in to "the wild" that reveal high-level operational vulnerabilities that might be shared amongst similar platforms, providing attackers with a so-called day zero attack that is cross-platform.

It is important to understand the broad overlap in the general construction of the major resource managers studied here, and expect that as vulnerabilities are found, reported, and fixed, that there should be overlap in the vulnerability space as well. This offers incentives for major operating systems to share findings with each other, to offer mutually beneficial security enhancements, but is likely to be complex to navigate in a commercial or competitive environment. However, for this study, it is valuable to observe that software systems in general overlap a great deal in the design patterns employed, and hence can corporately suffer from classes of vulnerabilities that affect all platforms.

3 Vulnerabilities

Not all attacks rely on vulnerabilities in the source code and as such not all attacks directly relate to open source software security. Many attacks rely on humans as the weak link, or at least rely in part on humans to help. In addition, there are other types of cybersecurity attacks that can be executed even on perfectly secure software. We classify a variety of popular attacks into source-dependent and source-independent and examine them in the following sections.

3.1 Source-Dependent Attacks

Open source software has not been a major target of security attacks, and this is generally thought to be due to the predominance of Windows as a platform. The number of Linux security attacks is increasing, however, due to increased popularity of Linux. There are currently more than 100 viruses that target Linux [2] and this number is constantly rising. There are also other attacks against Linux, including worms, Trojans, denial of service (DOS) attacks and rootkits.

Many Linux servers have been attacked and attacks are increasing [3], because many servers on the Internet run Linux. The main reason cited for the vulnerability of Linux servers is due to poor server administration, and not due to inherent security vulnerabilities in the software (Ibid). The complexity of administering a Linux server is also cited as one of the reasons to use a Windows server instead of Linux [4].

This raises another source of insecurities, which is the complexity of administrating the server. Poor administration leads to insecurities. Different distributions of Linux leave multiple ports open, increasing the vulnerability of the system to attacks. A default Windows installation may leave ports open, but because it is generally easier to administer, it is easier to close unwanted services than on Linux. Systems that offer better ease of administration are therefore preferred, but this issue is not inherent to source distribution philosophies, and so is beyond the scope of this work. It is important to note that ease of administration does not mean that one OS (whether open or closed source) is more or less secure than another. An system administrator expert in any OS will know how to best reduce vulnerabilities and secure the server.

There is one major way in which Linux and Windows are administered, which is that Linux administrators generally don't run as "root", instead running as a

normal user account, and use "setuid" or "sudo" to execute commands as root. Windows administrators, on the other hand, typically run using Administrator mode all the time. While this is not exactly an open vs. closed source issue, it can help explain why there might be more successful attacks on Windows than Linux.

Linux viruses often infect ELF (Executable and Linkable Format) files, which are common on Linux systems. Other attacks use shell scripts, which are often compatible across platforms and distributions, allowing them to spread more widely than a binary file. The sophistication of Linux viruses has also increased. For example, a recent virus, W32/Etap.d is a polymorphic virus, making it difficult to detect. A variant of this virus is able to infect Windows portable executable files in addition to Linux files.

Because open source code is often shared among different open source software (e.g. different Linux distributions or applications), some vulnerabilities can exist on multiple open source software. Apache is one example. The Linux/Slapper worm makes use of a known vulnerability in the Open SSL library to infect Apache web servers. In addition, the Linux/Adore.worm uses a random port scan to identify systems that contain a root access vulnerability on Linux servers. Even though there is only one vulnerability, multiple operating systems are affected.

Because open source software comes in many different flavors (e.g. the many varieties and distributions of Linux), it is more difficult to coordinate a simultaneous patch for all distributions. Since most attacks are made after both the vulnerability and patch have been made known publicly, but not all distributions have patches available, some distributions will be left vulnerable to attacks while others can be patched. This is a clear disadvantage of the distributed nature of open source software development. However, by the same argument, it is more

difficult for attackers to target multiple distributions simultaneously, so there are both positive and negative aspects to open source software distributions.

3.1.1 Buffer Overflow

Buffer overflows are easily found via automated tools that perform static code analysis; since this type of attack is specifically related to the structure and symbols in a given program's construction, we state that buffer overflows are source level attacks. As such, they are quickly detectable from any body of visible source code. Hence, any open source project with buffer overflows vulnerabilities in its code base will be susceptible to such an attack, and adversaries could use automated tools to perform a vulnerability analysis. However, these same tools are available during the lifecycle of the software under construction, and should be a routine part of any security analysis and walkthrough the code receives. In fact, many modern development environments will automatically warn developers if they have used legacy code constructs that could be subject to overflow attacks, and newer programming languages seek to disallow the types of memory structure declarations that allow programmers to overrun memory spaces.

An example of a buffer overflow exploit is the infamous Blaster worm released in 2003 exploited a vulnerability in the Windows DCOM service. The Blaster worm spreads without any user participation, such as opening an e-mail or visiting a web site. [5]

3.1.2 SQL Injection

SQL injection attacks generally follow the same strategy – find a web server that advertises database services and look for an interface, such as a username and password login webpage, that accepts client-side input. By providing carefully constructed input to the interface, the SQL database or even the server it is

running on can can be compromised. This attack is analogous to the buffer overflow attack for remote code execution in that it exploits failure to properly verify and limit input data. As such, it is a source-dependent attack.

3.1.3 Patch Reverse Engineering

A so-called "day zero" attack is the result of reverse engineering a recently released software update, in an effort to discover and exploit the problem the patch is supposed to fix. The patch itself serves as a blueprint describing a real vulnerability that most computers (unpatched thus unprotected) are susceptible to. The "day zero" moniker is derived from the fact that the best time to exploit the greatest number of computers is immediately following the patch release because the exploit becomes less valuable as the patch is applied to an increasing number of systems. In preparation for this style of time sensitive attack, tools are generated by attackers that can automatically generate a working exploit, given a patch as input.

Day zero attacks are code level attacks, and a disadvantage to open source software. Interesting to note here is that "closed" source software suffer from this exact class of attacks, too. The subtlety lies in the fact that the differences in machine code between patched and unpatched files can be reverse engineered to give an attacker visibility into the original security flaw, regardless of whether the source code is available. So, in this class of attacks, both open and closed source software are at risk.

3.2 Source-Independent Attacks

A large set of attacks do not rely at all on security flaws in source code, and thus neither open source or proprietary software is safer with respect to such attacks. In this section we describe some of these attacks to demonstrate that for these

classes, systems are equally vulnerable regardless of the software development philosophy employed when creating the operating system and other software on it.

3.2.1 User Participation

Many types of data theft (both corporate data and phishing for consumer data), spyware, viruses, and other attacks require some participation by a system user. Often these attacks just rely on the user executing a program or providing data to a party they should not. Often such attacks exploit the ignorance or inattention of the user. For example, many users will click "Yes" on any dialog that pops up just to make it go away.[6] Traditional viruses propogated when users transferred files between computers, often by floppy disk, and ran an infected application. Such viruses do not rely on security flaws in the applications that they infect.

If a user is logged into a system with root or administrator priveleges, any application that is executed during that session can also run with such privileges. This includes installing drivers, patching the kernel, etc. Other attacks may rely on user participation to exploit vulnerabilities, such as those that permit elevation of privileges. For example if there is such a security flaw in a part of an operating system that is not accessible via the network, an attack may require an unprivileged user to launch a program that implements the exploit. The vulnerability exploited as part of the Red Team exercise is one such example. The target application was not accessible via the network, but once the Red Team logged on with a limited user account it was able to execute the program and elevate the accounts privileges.

Attacks sent via email attachment are a common example of those that require user participation. An adversary creates a script or program that generates "socially-engineered" emails that entice the user to open the attached file.

The infamous "Love Letter" (also known as "I Love You")[7] and MyDoom[8] worms are examples of such attacks. While these worms attacked a specific operating system, they were dependent on user participation rather than exploits to spread.

3.2.2 Brute Force Attacks

Brute force attacks, such as dictionary password attacks on systems that permit remote access, are also independent of source bugs. In the case of dictionary password attacks, adversaries repeatedly try to log into a system (often as root or administrator) until they are successful. No code vulnerability is necessary to execute such an attack and therefore software based on both open and proprietary source is susceptible.

3.2.3 Protocol Vulnerabilities.

Some industry standard protocols have vulnerabilities inherent in the protocol itself or its dependencies. Since all software that supports the protocol is implementing the same protocol, all such software is equally susceptible. The Kerberos "Man-in-the-Middle" vulnerability is one example. Any standard implementaiton of Kerberos could be exploited regardless of source code availability. Such protocol vulnerabilities may be more worrisome than software vulnerabilities because the potential set of targets is larger as it includes many different platforms.

3.2.4 Physical Intrusion and Inside Jobs

While clearly the most pedestrian of the techniques surveyed thus far, no less effective is the physical intruder or an "inside job" executed by someone with access. In addition to offering immediate access to any system on the Internet

remote attacks also offer an additional level of anonymity and security for adversaries. In contrast, the simplicity of a physical attack is alluring for some, especially when network security is tight and a specific target is desired. When an adversary has physical access, the software implementation philosophy rarely matters.

3.3 Frequency of Attacks

The reported numbers of cyber attacks for all operating systems has grown steadily and, during some outbreaks, explosively, in the past decade and beyond. CERT, the coordination center for the Computer Emergency Response Team at Carnegie Mellon, has reported more than three hundred thousand incidents (where each incident could involve whole networks) since 1990. CERT reports that, given the growing use of automated attack tools, attacks against networked systems have become so frequent that "counts of the number of incidents reported provide little information with regard to assessing the scope and impact of attacks. Therefore, as of 2004, [CERT] will no longer publish the number of incidents reported" [9] and in place of sheer numbers, CERT will work to develop a more meaningful metric. However, we are able, at the very least, to note an increase in interest in and use of cyber attacks over the past decade and a half, so we include figure one. The growth in the number of attacks might seem to suggest multiple things, but we must take care not to derive too many conclusions; these statistics are for all operating systems and each incident could represent a completely compromised network or simply one computer. The increase in automation of attacks would help account for some of the explosive growth, and the increase in personal computer sales from 1988 to today provides for significantly more possible targets to attack (all sharing a similar base of operating systems and applications).

3.4 Understanding Attack Frequency

Many observers agree that a significant reason why Windows has been subject to more attacks than Linux is because it is a much more popular operating system [10]. In 2004, Linux market share globally was only about 3% [11], about the same as Mac OS, leaving Windows with most of the remaining 94%. Given this data, replacing closed source software with open source would not solve the problem – hackers would simply move to the next major target. As we will shortly see, Linux viruses exist, but in much fewer numbers than for Windows. In fact, some vulnerabilities have been left unpatched in Debian Linux because they did not feel the risk of an attack was significant enough to fix (Ibid).

4 Security Analysis

4.1 Vulnerability Discovery

Currently, it is mainly left up to the open source community to find and fix security issues in open source software. However, with the growing endorsement and investment of open source software (e.g. Linux) from large corporations such as IBM and Novell, more people who are security experts (who are employed by those corporations) will be finding and fixing security issues. Thus, we can expect the security of open source software to increase in the next few years. This will reduce any difference that exists in the security of open and closed source software.

For closed source software, security processes are now commonplace (e.g. at Microsoft) to attempt to reduce the number of vulnerabilities that exist in the software. These companies employ software security experts and even have security teams set up such as Microsoft's MSRC [12]. These processes enable companies to fix problems before they are discovered by customers, or a

hacker. As with open source software, security bug reports also come from the community.

4.2 Security Tools

4.2.1 Threat Modeling and Exploit Classification

Threat modeling is a method commonly used to find and address security threats in software. It is basically a brainstorming process that can be summarized as follows: A list of all inputs into the program is created. A list of all vulnerabilities in the program is created. These may be external dependencies (for example, APIs which the programmer has no control over) or other vulnerabilities which are not easy to eliminate.

Vulnerabilities are typically classified according to the STRIDE classification system. STRIDE is an acronym which stands for: Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privileges. They each refer to different classes of threats. "Spoofing identity" refers to obtaining access and use of another person's authentication information, such as username and password. "Tampering with data" refers to modifying data without the owner's permission. "Repudiation" refers to the ability to keep track what actions were performed, and who performed them. "Information" disclosure refers to obtaining information without having permission to access it. "Denial of service" refers to attempts to prevent legitimate users from using a service or system. "Elevation of privileges" refers to where an underprivileged user obtains higher-privileged access.

The STRIDE model was developed by Microsoft and is used by developers to identify security issues in their code.

After categorizing each of the vulnerabilities; for each of the inputs into the program, ways in which the inputs could threaten the vulnerabilities are listed.

For example, a program that sends data across a network might have a threat that the data could get tampered with as it is being sent across the network. Finally, mitigations for each threat are listed. A mitigation for the network example above could be to encrypt the data before sending it using a strong encryption method, or to use checksums. If a program is too big for all inputs to be listed at once, it can be broken down into features and subfeatures, and a threat model performed on each of these.

4.2.2 Source Code Scanners

Source code scanners (such as Flawfinder, RATS and ITS4) exist for both open and closed source software, and help create more secure code by finding common security issues in source code and often suggesting more secure code that could be used instead [13].

Both Flawfinder and RATS (Rough Auditing Tool for Security) are free, while ITS4 (It's The Software Stupid Source Scanner) charges a fee for its use. All three tools find problems with using functions such as printf() with variable length strings, gettext libraries and their use in internationalization. All three are configurable, allowing different levels out output (depending on the priority of the issue found) and ignorable lines. Flawfinder is the fastest of the three, while RATS finds the most errors (Ibid). One problem with all three tools is that they do not do any preprocessing, so some mistakes can go unnoticed (for example, if you create a macro for the printf function). ITS4 does not give as useful output as the other two tools, as it does not suggest secure alternatives.

A commercial tool, Microsoft Visual Studio 2005, includes two source code security scanners, one called FXCop, which scans managed, .NET Framework code; and one called PREfast, which scans regular C/C++ code [14]. These tools were used internally and have been tested within Microsoft before becoming part of Visual Studio. They both scan for a large number of potential errors,

which can be selected individually from a settings dialog. Suggestions on how to fix the code are also given.

The main limitation of source code scanners is that as a general-purpose automated tool, it will never be as thorough as a manual code review. A good suggestion is to use a code scanner as a pre-pass before the programmer does a manual audit of the code. It is also important for the programmer to understand what each API used does. For example, some libraries may be using unsafe APIs in them, which a code scanner would not find (unless the source for them is scanned as well).

Internally, Microsoft uses another tool, called PREfix (Ibid) which takes much longer but searches for even more errors. Because the tool is internal, Microsoft here has an advantage over open source code. But as mentioned earlier, as more large corporations invest in open source software, these advantages are expected to become less relevant in the near future.

4.3 Security Reviews

Closed source development is generally more organized than open source development, and thus there are more processes involved. One such process is the "security review", which most major software development companies use. These typically include Threat Modeling and running security tools statically on the code or during runtime. For example, Microsoft does a security pass at least once during each product development cycle. During a security pass, relevant source code is reviewed for security issues, code analysis tools are run on the source code and the bugs fixed, and tests, which try to find various security issues, are run on the software.

Open source development, being more distributed and less conventional, does not always go through a security process. Security review methods that can be

used effectively in open source projects include running code scanners (such as Flawfinder, RATS, and ITS4) and the fact that there are a large number of people looking at the code. However, the people looking at the code may not be experts or understand the code fully, which could let more security bugs go unnoticed. In addition, the tools available to large corporations are often more advanced than those available to the open source community. Furthermore, corporations - as with individuals - have access to all the tools in the open source community. The open source community may also be limited as to the APIs that can be used, where more secure APIs exist commercially.

While open source may have traditionally been a less organized "grass roots" effort, that is not always the case, especially for the major open source projects. Over the past decade, corporations have invested billions of dollars in open source software - mostly Linux. [15][16] IBM, Intel, Red Hat, and many other companies have full time employees contributing to Linux. In addition, Open Source Development Labs (OSDL) acts as "the central body dedicated to accelerating the use of Linux for enterprise computing."[17]

It is worth noting that around 95% of software bugs are caused by 19 'common, well-understood' programming errors" [18].

5 Socioeconomic Effects

The availability of the cource code for open source software has additional effects on the ecosystem of such software beyond just source code availability. These effects include the speed at which security fixes are available and the ability to customize security, to name a few.

5.1 Business Decisions

Some security vulnerabilities are worsened by business decisions that are unrelated to the product's source code availability. For example, Microsoft made the decision to enable the DCOM service by default in Windows XP despite the low utilization of this future in the user base. [19] While the vulnerabilities discovered in DCOM was a code flaw, the Blaster worm that exploited it would not have spread nearly as quickly or broadly as it did if DCOM had been disabled by default.

The Code Red and Nimda worms took advantage of a similar business decision that left vulnerable services enabled by default. Default installations for Microsoft's Internet Information Services (IIS) web server (versions 4.0 and 5.0) included the Microsoft Indexing Service. As a result, a vulnerability in the Indexing Service could be exploited on most systems using the IIS web server regardless of whether the system was actually running the Indexing Service. [20] Code Red alone is estimated to have caused billions of dollars in lost productivity. [21] The impact would have likely been reduced had only systems that needed it had the Indexing Service not been installed on servers that were not using it.

The decisions or lack of awareness of customers can also have an impact on the success of exploits. Once a patch is available, it is important for customers to apply them to their systems. A patch for the DCOM vulnerability was available for nearly a month before Blaster was released [22], and Nimda exploited the same vulnerability (for which a patch was available) as Code Red had a few weeks earlier. [21] (Note: The average delay between the publishing of a vulnerability or patch and the release of malicious exploits has decreased significantly since these worms were released.)

If different decisions - both by software vendors and customers - had been

made, some of the most damaging and infamous worms in the Internet's history may have been merely minor issues or not created at all because of the low potential return on investment for the author. Both open source and proprietary software products are susceptible to business decisions worsening the impact of or exposing code flaws. For example, one need not see the source code for DCOM to understand that this service was a potential vulnerability. They are also both susceptible to customer ignorance, apathy, or delay in applying patches for known vulnerabilities.

5.2 Accountability and Support

Accountability is an issue that often comes up when large organizations are considering adopting open source software. For example, a recent editorial in Federal Computer Week [23] says "CIOs considering a move to open-source software need someone to hold accountable – someone who has the resources to address any problems that occur." They want to know that they will have support available when they have problems. This is especially important for security issues, because they can be very costly to corporations, as shown by the numerous worms and viruses that have spread recently.

When a company buys software from Microsoft (as an example of a closed source system), they expect Microsoft has a level of accountability. The EULA (End User License Agreement) users agree to in order to use the software does not actually provide any accountability, but the reputation of software gives a company a level or accountability. If the software does not work, people will not use it, so it is in Microsoft's interests to fix security problems as they are discovered. In fact, this is what happens, as attested to by the numerous security updates that Microsoft releases.

When a company buys a Linux solution from IBM (as an example of an open

source system), there is also no accountability provided by IBM, but because they do not own the software (Linux software is distributed under the GNU Public License), there is less loss of reputation for any security issues that are found. IBM will provide setup support, but will not fix any bugs or security issues. These issues are left up to the Linux distribution's open source community to fix. Because there is no company that can be immediately contacted or responsible for fixing the issue, some corporations will not use open source software.

5.3 Patch and Fix Distribution

Microsoft releases regular security patches which they call "critical updates". Apple also releases monthly (or, as necessary) security updates for Mac OS X. Users have the option to be alerted when security patches are available, and then let them download and install the patch manually; or download automatically and let them install manually; or automatically both download and install the patch. One problem is that users typically need to reboot after installing a security patch, which may delay installation of the patch since many users do not wish to be forced to reboot their computer, instead preferring to keep their applications and documents open for days at a time. In this situation, the computer is vulnerable until it is rebooted, even though the patch has been downloaded and installed. This problem exists less on Linux, where reboots are not required as frequently [24].

Several major Linux distributions (e.g. Debian, Red Hat) also have automated security updates, which have similar features to Microsoft's security updates. While the patch mechanism is not an open source vs. closed source issue, it shows that the open source community has the resources to adequately issue patches and notify users of patches.

5.4 Availability of Security Fixes

When proprietary software vendors fix security flaws, they must validate it and get it into a release or patch. This process could take up to a year or more in the worst case. In some cases, the company may decide not to fix the flaw at all. In the open source community, however, end users could apply a fix as soon as it is implemented. This process is complicated by the fact that vendors must avoid incompatibilities with their customer base. Microsoft says the following regarding this issue.

> Microsoft products run on thousands of different manufacturers' hardware, in millions of different configurations, and in conjunction with countless other applications. Our patches must operate correctly on every single machine. This is a significant engineering challenge under any conditions, but it is even more difficult when details of a vulnerability have been made public before a patch can be developed. In such cases, speed must become our primary consideration, in order to protect our customers against malicious users who would exploit the vulnerability. [25]

Furthermore, significant new security features may not be available to users until the next major release or service pack. Software is often only released every couple or few years, so significant new security features may be withheld from users for years regardless of how long the feature took to implement.

There are also vendors that package open source software and must follow the same steps and be just as careful when releasing patches or other software updates. However, if an end user really wants the fix and is willing to accept the risk or test their configuration (i.e. in an IT department), they can get the fix immediately. Even if an open source vendor does not fix a security flaw, if it is important enough to end users, someone will likely implement a fix. In

addition, its possible that some group or vendor will release a distribution with the feature. Because there are many different distributions of Linux, there is competition between the makers of these distributions which are based mostly on the same code base. This means that once one distribution has a feature, there could be pressure on other distributions to add it as well for fear of losing customers. Regardless, users could always switch to another distribution – while switching distributions can be a hassle, it is much simpler than changing operating systems all together (i.e. Windows to Linux).

5.5 Custom Software and Configurations

In addition to permitting end users to apply security fixes as early as they wish, open source software also allows end users to make their own modifications to improve security. Anyone with the right knowledge and skills can make a derivative or custom version of open source software. One might choose to do so to limit exposure or increase security. Linux and Windows each contain tens of millions of lines of code.[26] Security vulnerabilities could be waiting in nearly any portion of that code, yet most users only need a minority of the code. As noted earlier, the Blaster, Code Red, and Nimda worms took advantage of vulnerabilities in features that many Windows users did not need. By compiling and installing only features that one really needs, there is less code with potential bugs and less code to review for security flaws if one chooses to. Custom builds do create their own potential vulnerabilities since the changes have not been reviewed by or tested by the wider community. However, as mentioned earlier, adversaries are more likely to go after more widely distributed software, unless of course you are a high value target. Custom builds also require that the maintaner merge applicable security fixes into the custom build. One solution to both issues is to submit the changes or derivative back to the community.

Other benefits of the ability for end users to customize software include the ability to add features or security updates to an existing version of software without being forced to upgrade to a new version, which may contain new vulnerabilities. The opposite can happen as well – features or support may be removed from a newer more secure version of software so users continue to run an older less secure version. This often occurs when hardware vendors make a business decision not to provide support (drivers) for older hardware on new versions of Windows. Consumers must then decide whether to pay to replace their hardware so that they can upgrade Windows or continue to run an older version of Windows. If the driver were open source and there was enough interest, someone may update the driver and provide it to the general public. In addition, if a open source software vendor does not think implementing a customer's request makes business sense, the customer or an organization of customers can modify the software themselves to suite there needs.

Security Enhanced Linux (SE Linux) is one such example that grew out of the United States National Security Agency's (NSA) frustration with computer-security weaknesses in the late 1990s. When proprietary software vendors rejected the idea of investing large amounts in extra-secure systems, the NSA began implementing its own derivative of Linux with an advanced security architecture. The agency then submitted SE Linux for inclusion in the Linux kernel and SE Linux eventually made its way into commercial products.[27][28]

Rootkits, such as the recent one found on some Sony music CDs, provide a us with a specific example of a modification a user may wish to make. The XCP rootkit on these CDs relies on the ability to patch functions in the system call table. [29] While Microsoft acknowledges that allowing patching of the kernel is undesirable, they have decided to continue to permit it in 32-bit versions of Windows because preventing it "would break compatibility for a significant

amount of released software." [30] If Windows were open source, end users would be able to decide for themselves whether protection from such rootkits or compatibility with software that "can interfere with other software and affect the stability of the operating system" [30] is more important to them. The answer would vary among end users, but at least they would have the option. Instead, end users that are concerned and knowledgeable enough are left in an ongoing battle to detect whether a rootkit has been installed on their system.

5.6 Security Fixes for End-of-Lifed Software

As with many other industries, software vendors like to end-of-life support for older software. For various reasons, customers may wish to or have business needs to continue using software beyond its supported timeframe. When proprietary software is no longer supported, these customers may be left vulnerable to new exploits. With open source, however, customers could make their own security fixes for security flaws indefinitely even if the original author or vendor is no longer releasing security updates. The same benefits apply if a vendor goes out of business.

Although not directly related to cyber security, the *Y2K* phenomenon demonstrates the need for source code. Countless software applications had to be replaced or updated to support four-digit years. Many companies decided to update their decades-old software rather than migrate to new hardware and software. Fortunately for many of them, the old software had been developed in-house, in which case they had the source code, or by vendors that provided year 2000-complian updates. Companies with commercial software for which no source or updates were available may have had to make different - potentially more costly or risky - decisions.

While open source software can help alleviate such scenarios, proprietary

software can also offer such cyber security blankets. Potential solutions for proprietary software include licensing the source code for specific use or putting it into escrow. Microsoft has even provided Windows source code to some customers, such as large governments.[31]

6 Conclusion

While open source makes code visible, we have found that source visibility does little to increase the security risks posed to a given project, in most cases. Security analysts report that, in some situations, source level visibility can even assists in the convergence of projects to stable and secure states.[32]

The current scale and prevalence of closed source systems relative to open source makes discerning meaningful conclusions from attack frequency alone difficult. It is expected that hackers will attack those targets offering superior economies of scale, but this does not preclude the possibility of closed source systems being more secure based on the sheer number of assaults mounted against such a system. If the operating system a hacker chooses to attack is rarely deployed, then the development of a corresponding exploit for that machine is of relatively little value. However, an exploit that can successfully compromise a wide base of systems is valuable indeed; we conclude only that popular systems are popular for attackers.

Closed source advocates will frequently cite the structure and adaptive remote development techniques as being too ad hoc and thus insufficient, not capable of uniform and coordinated evaluations or reviews by professionals. Open source is not a community of hobbyists, but rather a diverse grouping of professionals and full time industry experts from international corporations such as Intel, IBM, and RedHat. The closed source software lifecycle, while necessarily structurally different due to topology differences between contributers, has not

to date been shown in any quantative fashion to be inherently less secure.

Open source advocates frequently note their system to be more secure based on attack frequencies, as well as other observations that frequently are supported by strong opinions more than available data. In this research, we have been unable to demonstrate greater security through obsecurity or cloaking, and thus unable to assert that closed source offers additional protections or tangible security features we can leverage in pursuit of a more secure system.

The ability to read the source in and of itself offers little additional incentive for hackers to specifically target open source projects. If the secrets in software were completely containable (possibly through encryption), then perhaps a "security through obscurity" argument would apply more concretely; instead, however, since source code can be reverse engineered (to various levels of readability or success), the value of hiding the implementation is diminished. Further, exposing the code has side-effects that can be beneficial (such as unsolicited security reviews or software reuse), while cloaked source has no such counterparts. Finally, code leaks are somewhat common and difficult to prevent[33], so it is likely that any closed source project has been "opened" at some point, diminishing the value of the information hiding further. In addition, as described in this paper, many security attacks are independent of the source code, so neither open source or proprietary software is less secure.

In conclusion, open source does not pose any significant barriers to security, but rather reinforces sound security practices by involving many people that expose bugs quickly, and offers side-effects that provide customers and the community with concrete examples of reusable, secure, and working code.

References

[1] D. A. Wheeler, "Secure programming for linux and unix howto." http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/open-source-security.html, March 2003.

[2] "Linux - the new target for threats?." http://www.gadget.co.za/pebble.asp?relid=362, 2005. Gadget.co.za.

[3] M. Broersma, "Hack attacks on linux on the rise." http://news.com.com/2100-1001-943911.html?tag=cd_mh, July 2002. News.com.

[4] E. Hurley, "Scrap iis?." http://search390.techtarget.com/tip/1,289483,sid10_gci779497,00.html, November 2001. Tech Target.

[5] S. Bekker, "Blaster worm exploits rpc dcom vulnerability." http://redmondmag.com/news/article.asp?EditorialsID=5912, August 2003.

[6] "The default answer to every dialog box is 'cancel'." http://blogs.msdn.com/oldnewthing/archive/2003/09/01/54734.aspx, September 2003.

[7] "Cert advisory ca-2000-04 love letter worm." http://www.cert.org/advisories/CA-2000-04.html, May 2000. CERT/CC.

[8] S. Harris, "Cyber alert system debuts as super worm spreads." http://www.govexec.com/dailyfed/0104/012804h1.htm, January 2004. National Journal Group.

[9] "Cert/cc statistics 1988-2005." http://www.cert.org/stats/cert_stats.html, 2005. CERT Coordination Center.

[10] R. Gedda, "Open source security 'not good enough'." http://www.cio.com.au/index.php/id;1607539824;fp;512;fpid;1435609079, 2004. CIO.

[11] G. Keizer, "Linux to ring up $35 billion by 2008." http://www.techweb.com/wire/showArticle.jhtml?articleID=55800522, 2004. TechWeb.

[12] "Microsoft security." http://www.microsoft.com/security/default.mspx, 2005. Microsoft, Inc.

[13] J. Nazaario, "Source code scanners for better code." http://www.linuxjournal.com/article/5673, 2002. Linux Journal.

[14] J. Udell, "Source code analysis breaks new ground." http://www.infoworld.com/article/04/10/29/44FEsource_1.html, October 2004. Info World.

[15] J. Wilcox, "Ibm to spend $1 billion on linux in 2001." http://news.com.com/2100-1001-249750.html, December 2000. CNET News.com.

[16] "Ibm's open-source army." http://www.redherring.com/Article.aspx?a=13124&hed=IBM's+Open-source+Army, August 2005. Red Herring.

[17] "About osdl." http://www.osdl.org/about_osdl/. Open Source Development Labs (OSDL).

[18] http://www.acm.org/technews/articles/2004-6/1206m.html#item5.

[19] M. Liron, "Dcom windows xp: Do you need it?." http://www.updatexp.com/dcom-windows-xp.html, September 2003.

[20] R. Boyce, "Code red - iss buffer overflow." http://www.sans.org/resources/malwarefaq/code-red.php. The SANS Institute.

[21] E. Hurley, "Scrap iis?." http://search390.techtarget.com/tip/1,289483,sid10_gci779497,00.html, November 2001. TechTarget.

[22] "Ms03-026: Buffer overrun in rpc may allow code execution." http://support.microsoft.com/?kbid=823980, October 2005. Microsoft.

[23] "Reviewing the case against open source." http://www.fcw.com/article88468-04-04-05-Print, April 2005. FCW.com.

[24] "Security report: Windows vs linux." http://www.theregister.co.uk/security/security_report_windows_vs_linux/, 2004. The Register.

[25] "Acknowledgment policy for microsoft security bulletins." http://www.microsoft.com/technet/security/bulletin/policy.mspx, January 2000. Microsoft TechNet.

[26] D. A. Wheeler, "Counting source lines of code (sloc)." http://www.dwheeler.com/sloc/.

[27] D. Clark, "Defense, nsa move on 'open source' software development." http://www.govexec.com/dailyfed/0303/031703td2.htm, March 2003. National Journal's Technology Daily.

[28] M. S. Mimoso, "Red hat brings se linux to fedora." http://searchopensource.techtarget.com/originalContent/0,289142,sid39_gci957636,00.html, March 2004. SearchOpenSource.com.

[29] M. Russinovich, "Sony, rootkits and digital rights management gone too far." http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html, October 2005.

[30] "Patching policy for x64-based systems." http://www.microsoft.com/whdc/driver/kernel/64bitPatching.mspx, September 2004. Microsoft WHDC.

[31] D. Appleman, "Opening source: The ultimate customer security blanket." http://www.devx.com/opinion/Article/20513, March 2004.

[32] "Creating a national framework for cybersecurity." http://www.acm.org/usacm/PDF/CRS_cybersec.pdf, February 2005. acm.

[33] S. Kuchinskas, "Microsoft's loss not linux gain." http://www.internetnews.com/ent-news/article.php/3313241, February 2004. InternetNews.