The availability of source code in relation to timely response to security vulnerabilities

John Reinke [1] and Hossein Saiedian [2]

[1] Sprint, USA

[2]University of Kansas, USA
Department of EECS, School of Engineering
University of Kansas
Lawrence
KS 66045
e-mail: saiedian@eecs.ku.edu

Abstract

Once a vulnerability has been found in an application or service that runs on a computer connected to the Internet, fixing that exploit in a timely fashion is of the utmost importance. There are two parts to fixing vulnerability: a party acting on behalf of the application's vendor gives instructions to fix it or makes a patch available that can be downloaded; then someone using that information fixes the computer or application in question. This paper considers the effects of proprietary software versus non-proprietary software in determining the speed with which a security fix is made available, since this can minimize the amount of time that the computer system remains vulnerable.

1 Introduction

There is growing debate concerning the security advantages of proprietary software versus non-proprietary (open-source) software. Although the debate has existed for decades, it has grown more important in recent years as the public's awareness of computer security increases and as more services are now available via the Internet. Generally, there has been insufficient evidence to proclaim a winner [1]. While a strong argument can be made for each type of software in different cases, it really depends on the situation in consideration.

Since bugs inevitably occur in any kind of software [2], a top priority for system maintainers is to have a solid, usable solution available as soon as possible [3]. A security bug could put multiple computers or a complete network at risk, therefore all the critical data and services provided by those computers could be compromised. The sooner the problem is resolved, then the less time would-be intruders will have for gaining access, looking around, and performing malicious activities. If one type of software appears to be better at minimizing the time it takes to fix those problems, it could benefit many and outweigh other arguments against it.

2 Networks, applications and security

Most computers today are connected to other computers by some means of a network, i.e. a local network or even a dial-up connection to the Internet. It is through this connectivity that attackers are able to make their assault [4]. Through connectivity, attackers have a virtual smorgasbord or all-you-can eat buffet, since it brings access to all interconnected computers directly to them. While it might not be simple for them to gain access to those systems, the process can be automated.

2.1 Services and applications on computers connected to the Internet

The main targets for attackers are Internet services [5]. These are the applications that allow everyone to use e-mail, web browsers, chat services, and much more. In order to make those services reachable whenever they are needed, the applications must run continuously, usually for days, months and even years on end. Continuously running applications are usually referred to as servers or daemons.

Attackers like vulnerabilities in software running on servers because, when vulnerability exists, they will have continuous access to the

ELSEVIER

Computers & Security
Vol 22, No 8, pp 707-724, 2003
Copyright ©2003 Elsevier Ltd
Printed in Great Britain
All rights reserved
0167-4048/03

John Reinke

John Reinke is software developer and network engineer at Sprint Corporation in Overland Park, Kansas. He completed his Master's degree in Computer Science at the University of Kansas in 2002.Hossein Saiedian (PhD, Kansas State University, 1989) is currently a professor software engineering in the Department of Electrical Engineering and Computer Science at the University of Kansas. Professor Saiedian's primary area of research is software engineering and in particular models for quality software development, both technical and managerial ones. He is particularly interested in formal models and their integrations with semi-formal, graphical models. Professor Saiedian has over 100 publications in a variety of topics in software engineering, computer science, and computer security. His research in past have been supported by NSF as well as regional organizations. Professor Saiedian is a Senior member of IEEE.

exploitable service until it is fixed. By its very nature, daemons must make their services available to anyone on the Internet in order to fulfil their function. Therefore, computers running these services are typically connected to the Internet at all times. Of added incentive to attackers is the fact that service providers will probably not disable a service completely if there is a problem, because of the more immediate problem of damage to their business if they completely disable a service for which their customers are paying. Once again, this is why it is important for system administrators to have access to vulnerability fixes as soon as they are available.

Internet applications are the software programs that use the services described in the previous section. These applications include e-mail clients and web browsers, which are used by most households and businesses today. The significance of Internet applications is that they require connectivity to other computers on the Internet, even in the case of dial-up Internet service via phone lines. It doesn't matter much in the case of dial-up access, because some exploits work through malicious programs attached to an e-mail that is read while disconnected but sends messages to everyone in the address book when the connection is re-established. E-mail viruses like this can very quickly bring a network to a stand-still with all the extra traffic, in addition to the risk of damage to the data on infected computers.

Attackers commonly use exploits to Internet applications to gain control of computers. Once an application on the computer has been exploited, it is most likely that other applications or the operating system itself will become compromised. These compromised machines might then be used to make up an attack network — a virtual army of computers that can be used for a variety of attacks on other computers [6].

2.2 Proprietary software

The first type of software considered in this study is known as proprietary software. The best definition of proprietary software is software in which the source code is only available to the makers of the software. In most cases, the software is created by companies that rely on the sale of the software for their well-being, and there are other reasons why they don't provide the source code with the software.

The first reason for restricting access to the source code is because the company relies on sales of the software to support its business. If the general public had free access to the source code, then people could compile the code into the executable applications and share the executables. It would be very difficult for a company to continue if its main stream of income could be short-circuited like that, so software companies go to great lengths to protect the source code of their products.

A second reason for keeping source code protected is so that it cannot be changed to do something other than the intended purpose of the software. If there were multiple copies of the software available due to changes made by people with access to the source code, then the software company would not want to support or be held responsible for them. The company feels a need to control the features and quality of the software.

Another reason for keeping source code of applications from the public is for security. It is believed that, if no one can view the source code of the applications, then it will be harder to find exploits or security holes in the applications. This is often referred to as 'security through obscurity' and has been shown to have limited value, since it tends to create an inaccurate level of confidence in the software [7].

Where proprietary software is used

Proprietary software is used almost everywhere. Most operating systems and software for household computers is proprietary. This is also true of most office applications that are used in businesses. Many Internet services run on proprietary operating systems, including some versions of UNIX.

Sometimes also known as COTS (commercial off-the-shelf) software, it can be used for more specialized solutions. It is common for businesses to need software to perform specific tasks, and possible solutions include purchasing specialized software or building it "in-house". The decision often comes down to the overall cost for the solutions, the amount of time required to customize COTS software versus creating it from scratch, and the level to which each solution fills the requirements. The business has an advantage of access to source code if it chooses to build the software in-house, whereas in most cases the security of the COTS software will have to be trusted [8, 9].

Procedure for fixing a vulnerability with proprietary software

For the case where there is a vulnerability with proprietary software, there are typically two possible solutions. The first solution entails a type of patch. Usually, there will be an executable available to those that have a license to the software which will change the parts of the application that need to be fixed. Since it makes changes to the existing executable, the patch will often be freely available to download over the Internet. The patch will only be useful to those that already have a licensed copy of the software. Without the original executable, the patch is useless.

A second method for fixing a security problem is to make available a new version of the software. This is generally not preferred, since it would require distribution of the complete product. Complete products will most likely be too large to download easily, and the software company may decide to physically package the new version and send it through the postal service. This is not an effective solution if computers must remain vulnerable while the software is packaged and sent through the mail.

Another issue that affects the time it takes to fix security problems in proprietary software concerns the number of people working on the solution. Since the source code is only available internally to the software company, there is a limited number of people able to inspect the source code compared to the number if the general public had access to it.

2.3 Open-source software — source code availability

Open-source software (OSS) is software in which the program code is available to the person that chooses to use the software. While the software generally has no monetary cost, this does not always have to be the case.

There is a popular philosophy of Richard M. Stallman's [10] that the person that is using software has the right to have access to the source code. This is in contrast to one of the reasons why software companies protect the source code, namely control. Rather than the software company maintaining control over how their software is used, Stallman believes that it is the right of the user to know how the software is being used, and to be able to change it if they so choose. There is a variety of other reasons, and some are also security-related.

Where open-source software is used

One of the most common applications of open-source software is the Linux operating system and most of the tools and applications that come with it. Linux is very common for systems running Internet services, so it is very familiar to those in the security community.

Other examples of open-source software are apache (web server), sendmail (mail server), and bind (DNS server). This software is used on machines that host web sites, store or transfer e-mail, and direct requests on the Internet to the correct machines. A large share of the server systems on the Internet run these applications, although open-source software is growing increasingly common in desktop computers and development workstations as well.

Using source code to fix a vulnerability

There is a number of advantages to having access to the source code for an application that has a vulnerability. First of all, if the individuals using the software have an understanding of the vulnerability and the source code, then they may be able to locate the problem and fix it themselves. Secondly, since there are theoretically more people using the software than there are programmers that have created it, all those users of the software can help look for the problem in the source code, decreasing the time it takes to find a solution. Thirdly, if someone other than the vendor is able to find a solution, then they might either provide a patch for the application (to be applied to the source code) or publish instructions for users of the software to modify the source code and recompile the software with the fix themselves.

3 Data source

In order to evaluate the response times for proprietary and open-source software, a common and reputable source of data had to be found. Some of the requirements are that it is well known and trusted to those in the security field. It had to provide data in a timely matter and in an unbiased fashion, so no particular software vendor is favored. The sources considered are the BugTraq mailing list (available at SecurityFocus.com), the CERT Coordination Center (www.cert.org), and Incidents.org (from the SANS Institute).

The CERT Coordination Center was chosen as the source of data collected for this project. While there are undoubtedly many resources for security information, CERT/CC is considered by many to be the most central point of knowledge for all computer security-related events. It has a cleanly organized web site with clear definitions of the types of alerts available. It also has the unique distinction of being one of the first of its kind, and its creation was the result of desperate circumstances.

CERT/CC (the Computer Emergency Response Team Coordination Center) was not a project that came from years of careful planning. It was in fact created within days of the idea arising, due to the Internet events of 1988. One of the most defining moments for the Internet occurred on 2 November 1988. A program called a worm worked its way from machine to machine, by way of security holes in services running on those machines. It spread itself across the Internet, busying up networks, and slowing everything to a crawl. At the time, most organizations using the Internet were educational institutions and the government. However, it was the academic sites that were mainly responsible for finding out how the attack worked and how to stop it from spreading [11].

A lot was learnt from the worm, and everyone realized that changes needed to be made to better deal with the next attack. Some of the lessons learned were [11]:

> Sites that disconnected themselves from the Internet during the attack to prevent damage missed out on communications about the activities of the worm and the bug fixes, and were unable to help others analyze the worm. Additionally, they cut off communications for other sites whose e-mails needed to pass through their network to get to its destination.
>
> In case of emergencies like this, it is necessary to have a way to learn who to contact at other sites, and how they can be reached.

Groups needed to be formed, and it was interesting to see who became the leaders of these groups.

There was a lot of misinformation, which spread quickly.

Availability of source code was critical. The sites making the most progress in learning about the attack, as well as being able to defend themselves, had access to the source code for the vulnerable applications.

A conclusion reached afterwards [12] was that the only reason why the worm did not take longer to stop was the existence of the network of people working together able to jointly figure out what was happening and find a way of stopping it. It was recommended that a crisis center should be set up as a more formal and wider network. After the occurrence of another more limited attack later that November, DARPA (the Defense Advanced Research Projects Agency) decided to establish CERT (the Computer Emergency Response Team) as a central point of communication for Internet security issues, and part of the Software Engineering Institute, a non-academic unit of the Carnegie Mellon University.

3.1 Types of CERT/CC publications

There are three main types of security alerts and publications provided by the CERT Coordination Center: advisories, vulnerabilities, and incident notes. The number of occurrences of each in recent years is shown in Table 1, from data available at the CERT/CC web site [13]. The numbers, particularly the number of 'incidents' shown, demonstrate that they do not all deal with the same aspect of computer security 'incidents'.

CERT/CC incident notes

The best general description of a CERT/CC incident is 'The act of violating an explicit or implied security policy'. Since security policies vary greatly, the following guidelines are suggested by CERT/CC when considering whether to submit an incident report:

attempts to access a system without authorization, whether successful or not;

any interruption or denial of services;

usage of a system (without permission) to store or process data;

any modifications to a system's hardware, software or firmware, without the approval or knowledge of the owner.

Since so many incidents are reported, incidents involving life-or-death situations, threats to the infrastructure of the Internet (such as a root domain name server), and widely spread attacks are given a higher priority. Reporting incidents helps to identify new attacks and trends and increases the quality of security statistics, while providing data for better awareness and security fixes.

CERT/CC vulnerability notes

The next types of publications provided by the CERT/CC are vulnerability notes. Vulnerabilities are typically reported by the end-users or security experts that discover them and, ideally, have not been exploited by the bad guys yet. The notes typically describe security holes and critical bugs found in specific applications or implementations of protocols. Vulnerabilities must be taken through a process to ensure that the vendor is aware of the problem and working on it, especially if there are known exploits that are already spreading. If any fixes or workarounds are known, then they are listed as well.

CERT/CC advisories

CERT/CC advisories are vulnerabilities that are considered to be especially serious. These are

Table 1: CERT/CC Statistics

Year	1999	2000	2001	2002,Q3	Totals
Advisories	17	22	37	27	103
Incidents	9,859	21,756	56,658	73,359	161,632
Vulnerabilities	417	1,090	2,437	3,222	7,166

Table 2: Advisory Data Sample

Advisory	Description	Type	Public	Private	Fix	Info	Days
CA-2002-20	Multiple Vulnerabilities in CDE ToolTalk	PS	07/10/02	05/26/02		MV	
CA-2002-12	Format String Vulnerability in ISC DHCPD	OSS	05/08/02	03/24/02	04/09/02		16
CA-2002-11	Heap Overflow in Cachefs Daemon (cachefsd)	PS	05/06/02	03/22/02	05/24/02		63
CA-2002-09	Multiple vulnerabilities in Microsoft IIS	PS	04/11/02	02/25/02		MV	
CA-2002-07	Double Free Bug in zlib Compression Library	OSS	03/12/02	01/26/02	03/11/02		44
CA-2002-06	Vulnerabilities in Various Implementations of the RADIUS Protocol	MI	03/04/02	01/18/02			
CA-2001-26	Nimda Worm	PS	09/18/01	08/04/01		Prev	
CA-2001-23	Continued Threat of the "Code Red" Worm	PS	07/26/01	06/11/01		Prev	
CA-2001-19	"Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL	PS	07/19/01	06/04/01		Prev	
CA-2001-18	Multiple Vulnerabilities in Several Implementations of the Lightweight Directory Access Protocol (LDAP)	MI	07/16/01	06/01/01			
CA-2001-16	Oracle 8i contains buffer overflow in TNS listener	PS	07/03/01	05/19/01		N/A	
CA-2001-15	Buffer Overflow in Sun Solaris in.lpd Print Daemon	PS	06/29/01	05/15/01	08/30/01		107
CA-2001-13	Buffer Overflow In IIS Indexing Service DLL	PS	06/19/01	05/05/01	06/18/01		44
CA-2001-08	Multiple Vulnerabilities in Alcatel ADSL Modems NS	NS	04/10/01	02/24/01			
CA-2001-05	Exploitation of snmpXdmid	PS	03/30/01	02/13/01	08/30/01		198
CA-2001-03	VBS/OnTheFly (Anna Kournikova) Malicious Code	PS	02/12/01	12/29/00		None	
CA-2000-22	Input Validation Problems in LPRng	OSS	12/12/00	10/28/00	09/25/00		-33
CA-2000-07	Microsoft Oce 2000 UA ActiveX Control Incorrectly Marked "Safe for Scripting"	PS	05/24/00	04/09/00	05/12/00		33
CA-2000-04	Love Letter Worm	PS	05/04/00	03/20/00		None	
CA-2000-03	Continuing Compromises of DNS servers	OSS	04/26/00	03/12/00		None	
CA-1999-	17 Denial-of-Service Tools	MI	12/28/99	11/13/99		None	
CA-1999-12	Buffer Overflow in amd	OSS	09/16/99	08/02/99	09/09/99		38
CA-1999-04	Melissa Macro Virus	PS	03/27/99	02/10/99		None	

the items that are deemed by the CERT Coordination Center to be so important that everyone in the Internet community — including system administrators, software vendors, and end-users — should take notice immediately.

Advisories may comprise a vulnerability or multiple vulnerabilities of an application that are being exploited on a grand scale, such as worms and viruses. They might be wide-open security holes that are trivial for a beginner to exploit. In some cases, advisories might alert everyone to an attack on specific services on sites that have something in common, such as prominent web sites or any sites using the same version of defective software.

4 Data collection process

The types of CERT/CC alerts used for the collection of data are the CERT/CC advisories. These are based on vulnerabilities that present serious risk, so a certain amount of evaluation has already been performed to determine that a significant number of computers or networks are in danger. Occasionally, the advisory is linked to one or more vulnerability listings, and the additional information can provide more insight to the problem, solution, and other communities where the vulnerability was listed.

To reflect the current usage of proprietary and non-proprietary software, data was collected for CERT/CC advisories issued within the last four years. One of the security issues that has become commonplace in that time frame is the widespread adoption of broadband Internet connectivity, such as ADSL (asymmetric digital subscriber line) and cable modem services. This creates a new awareness to security, as household computers now have Internet connectivity 24 hours a day. Where security was wholly an issue for system administrators, it is now moving into the hands of the general public. Software vendors must consider that any application they create may be used while connected to the Internet, and it must not be the weak link that allows an attacker to compromise a computer. All significant bugs that are found must be fixed in the most timely manner possible.

The following process was applied for each CERT/CC advisory collected from the list of advisories (http://www.cert.org/advisories) at the CERT/CC web site. Sample results are included in Table 2, with an explanation of the abbreviations below.

In the first column of Table 2, CERT/CC advisories are shown in the format CA-YYYY-XX, where CA shows that this alert is a CERT/CC Advisory, YYYY indicates the four-digit year in which the advisory was issued, and XX identifies which advisory number it was within that year. The second column is the description, exactly as given in each advisory. The next column is the type of software that was determined for the advisory. The public date is when the advisory was released, and the private date is exactly 45 days prior to the public date, as described in section 4.4. The 'Fix' column contains the date that a solution was provided, if one was found for the advisory. The 'Info' column holds an abbreviation for special scenarios, if any were found. The final column shows the number of days needed by vendors to provide a solution for the vulnerability.

4.1 Determine the type of software

The software type could usually be determined from the web page for the individual advisory by looking at the vendor(s) listed. The type categorizes most software as either proprietary software or open-source software, as compared in this paper. The following is a list of each type of category with a brief description:

Multiple implementations (MI)

This indicates that the advisory affects software that is available from more than one vendor. Whether the implementations are all open-source, all proprietary, or any combination of the two, the fact that there is more than one application that needs to be fixed will effectively eliminate a single fix date for the vulnerability. Type MI advisories are not counted as either open-source or proprietary software.

Open-source software (OSS)
Software for which the source code is available with the application.

Proprietary software (PS)
Software for which the source code is not available to the end-user.

Non-software (NS)
This applies to vulnerabilities that cannot be tied directly to a software development environment.

4.2 Identify special scenarios

There is a number of special scenarios where no fix date is available. When one of these scenarios was identified, it was noted and no further information needed to be collected for the advisory. These scenarios and their identifiers are listed in the 'Info' column of Table 2.

Multiple vulnerabilities (MV)
In this case, the same protocol or application has numerous vulnerabilities that are addressed with a single advisory. Because not all the vulnerabilities were discovered or fixed on the same date, there was no single number that could represent the number of days that were needed for a fix to be made available. For this reason, no fix date is recorded for advisories with multiple vulnerabilities.

No information available (N/A)
This usually meant that not enough information was available at the CERT web site or the site of the vendor of the application. In some cases, the information appeared to be restricted from the general public.

No fix available (None)
There was no fix date available, either because there was no real fix that could be offered or because the vulnerability related to a 'feature' in the application that the vendor did not choose to disable. For example, it is usually up to the end-user to be cautious not to open e-mail with unknown attachments and to use anti-virus software where appropriate.

Previous advisory (Prev)
This indicates that there was a previous advisory for this issue, but it continued to be a major problem either because too few people took action to fix it or because the attack is aggressive and ongoing. There is no need to count the same solution twice, especially since it is likely to have been available before the follow-up advisory was announced.

4.3 Find the date on which a solution became available

The search for the date on which a solution or patch became available for the vulnerabilities involved a great deal of legwork. In a few cases, the date was available on the CERT/CC advisory page itself, often including a list of instructions for fixing or disabling the vulnerability. Most often, the information had to be found at the vendor's web site. Typically, if an advisory page was available at the vendor's site, then the actual patch or a new release of the application had to be found and the date found on which the file was created or made available. The longer ago an advisory was issued, the more likely it is that the vendor had redesigned their web site or changed the location of their posted security alerts. This requires old web links to be traced or found using a search engine capable of providing copies of pages that no longer exist. At other times, mailing list archives had to be searched or browsed for discussion of the vulnerability and availability of a solution.

4.4 Calculate the time needed to provide the solution

For every advisory where a solution was provided, the number of days required to provide the solution needed to be calculated.

This presented a challenge, since identifying the day when the vendors themselves learned of the vulnerabilities was nearly impossible. It would be in the best interest of the vendors of proprietary software to keep this information tightly protected from the public before a solution could be made available. Even after a solution was available, this information was still protected. Purportedly, this could be an issue for proprietary software makers if their customers learned after the fact how long it took for a solution to be made available, thus leaving their computer systems exposed. Even if the fix occurred within a reasonable amount of time, it could still affect a company financially.

Since it was difficult to find the original date on which vendors were notified, a reliable system had to be derived. From CERT/CC's disclosure policy [14], "All vulnerabilities reported to the CERT/CC will be disclosed to the public 45 days after the initial report, regardless of the existence or availability of patches or workarounds from affected vendors. Extenuating circumstances, such as active exploitation, threats of an especially serious (or trivial) nature, or situations that require changes to an established standard, may result in earlier or later disclosure." The policy also says, "Vulnerabilities reported to us will be forwarded to the affected vendors as soon as practical after we receive the report."

For both proprietary and open-source projects, counting back 45 days from the date on which the advisory was made public was found to be a more reliable system than attempting to discover these dates through research online. It was also found to be fairer, since much more of this information was found for the open-source projects than for the proprietary projects. Thus, the amount of time required to provide the solution was calculated by counting the number of days, starting 45 days prior to the date the advisory was published through the date on which a solution was first found to be made available.

Perhaps it should be added that collecting data from the time prior to that which we used might create different results, for the wrong reasons. Much of the earlier networked applications were open-source. This could increase the likelihood of alerts being for open-source. On the other hand, open-source application development had a head start in the world of security-aware programming, so those earlier dates might have benefited that side. Since one purpose of this paper is to provide some guidance or input into future software purchase/usage decisions, looking at data from too long ago might not best represent the abilities of software developers today. We also employed the same procedures to find those dates, for either type of software. To find the dates we first followed links from the CERT advisories to the project or company web site. A lot of open-source projects have a web site too, and it is common for the site to contain old news, security announcements, or mailing list archives. We basically searched for the earliest date we could find for the solution. This is the date that we used.

It is also true that CERT is not always the first to be notified when a vulnerability has been identified. They may only find out from the vendor. Therefore, the vendor may have known about the vulnerability for longer than the 45-day period. In most cases that we recall, it was first discovered by a security company or an individual who notified CERT and the company/developer at the same time. Sometimes it was mentioned on a security mailing list or on a list especially for that project. While it may be possible that some are first known by the vendor, it might also be possible that they are kept quiet and never become an advisory.

The data in Table 2 is solely there to provide examples of the different types of scenarios and bugs that we found. We picked things like the 'Nimda Worm' and 'Code Red', which readers might remember or be familiar with. One can also see that the advisory numbers skip many times. For example, it skips from CA-2001-26 to CA-2002-06, despite the fact that Table 1 shows that there were 37 advisories for 2001.

5 Analysis process

Once a system was developed to organize the collected information, the data needed to be analyzed to find in which direction it pointed and what could be learned from it. The data was stored in a spreadsheet, in order to automate calculations and also to provide a clean display of the advisories when studying them. The sections below refer to specific advisories by their CERT/CC advisory number, which can be found in Table 2.

The primary focus of this study is to find out if there is a security advantage to using proprietary or non-proprietary software with respect to the amount of time it takes to provide a solution to a vulnerability. To do this, the collected data was organized according to whether the security advisories relate to proprietary or non-proprietary software. Advisories that did not fit into those categories were also documented and analyzed to find any patterns. Additionally, cases where a solution was not found for the advisory were also categorized.

The categories describing the types of software are described below, together with the number of occurrences of each. The two most significant types for this study are proprietary software and open-source software. Table 3 shows a brief break-down of each by percentage.

Table 3: Software Types Data

Type	Percentage
Proprietary Software (PS)	56%
Open Source Software (OSS)	18%
Multiple Implementations (MI)	24%
Non-Software (NS)	2%

Proprietary software (PS)

Proprietary software made up the largest group of software types that were identified. It made up 56% of advisories that were classified, 57% of software-related advisories, and 76% of advisories determined as only proprietary or open-source. An example of an advisory classified under proprietary software is CA-2001-13 (Buffer Over flow in IIS Indexing Service DLL). In this vulnerability, a DLL in versions 4 and 5 of Microsoft's IIS web server contained a buffer overflow that would allow remote exploitation and execution of arbitrary code on the compromised machine.

Open-source software (OSS)

Open-source software was the sole type of software for 18% of all advisories, 18% of software-related advisories, and 24% of those classified as only proprietary or open-source software.

An example of an open-source software-related vulnerability is advisory CA-1999-12 (Buffer Over flow in amd). This vulnerability is a buffer overflow in the logging portion of the Berkeley Automounter Daemon. The daemon, which allows automatic mounting of devices when files on those devices are requested, can allow arbitrary code to be executed as the user running the amd — usually root (the administrative user).

Multiple implementations (MI)

Cases where multiple implementations existed for a vulnerability made up 24% of the overall advisories researched.

Advisories that found vulnerabilities within different vendor's version of the same type of software could not be counted easily. In some cases, the implementations occurred in both open-source and proprietary software, so they could not be counted as either type of software. Further, typically they could not be counted as one item for each type, because the number of occurrences did not split evenly between the two types. In other cases, even if all the occurrences appeared to belong exclusively to either open-source or proprietary software, many other variables did not match. These variables included not having the same date on which a fix for some implementations was available, even if it was based on the same code. For the proprietary software, it appeared that the same closed-source library might have been used for different implementations, but the vendors normally made their own solutions available. It was the collision of these variables that made advisories based on multiple implementations an unreliable source for data collection with respect to proprietary versus open-source software.

An example of multiple implementation of a vulnerable piece of software is CA-2001-18 (Multiple Vulnerabilities in Several Implementations of the Lightweight Directory Access Protocol (LDAP)). Multiple implementations of the LDAP protocol (including open-source and proprietary implementations) contain vulnerabilities, so a solution cannot be credited fairly to either type.

Non-software vulnerabilities (NS)

The final type of alert that was identified could not be tied specifically to a method of software development. Even if the problem developed within a proprietary device or environment, there would have to be an equal chance of the same device or situation occurring within an open-source environment to allow these vulnerabilities to be counted. Overall, non-software vulnerabilities only accounted for 2% of the overall CERT/CC alerts issued within the data collection.

An example of a non-software vulnerability is CA-2001-08 (Multiple Vulnerabilities in Alcatel ADSL Modems). The modems allow TFTP (trivial file transfer protocol) from the LAN side of the modem for firmware updates to be installed. However, there is a bug that could allow remote exploitation of this service by forging information in data packets sent from the outside of the LAN that would bounce back to the modem from a machine on the local network. Additionally, the default password for administration of the modem was an empty password, which makes exploitation especially easy for anyone with physical access to the modem.

5.1 Identification of special scenarios

Special scenarios are cases where a single solution date was not found to be available. The significance of these scenarios is that they prevented the identification of a date when a solution was available for the related vulnerabilities. While it may seem that not many vulnerabilities fit into this area, when combined they account for one third of all the vulnerabilities studied. Also, it is important to analyze this information to find any patterns that might relate to why open-source or proprietary software vulnerabilities were not counted as fixed.

Table 4 provides an overview of the number of special scenarios that were found. The percentages listed are for the overall items identified for the software type at the top of that column. For example, of all the advisories determined to be open-source software, 24% were found to contain multiple vulnerabilities. The last column compares each scenario type with the total number of vulnerabilities as a whole. Because no non-software-type alerts fell under any of the four scenario types, a relation between the scenario types and non-software vulnerabilities will not be studied.

Table 4: Special Scenario Occurrences

Scenario Type	PS	OSS	MI	NS	Overall
Previous Advisory (Prev)	9%	0%	9%	0%	7%
Multiple Vulnerabilities (MV)	17%	24%	4%	0%	15%
No Solution (None)	11%	6%	13%	0%	10%
Data Not Available (N/A)	2%	0%	0%	0%	1%

Previously existing vulnerabilities/solutions (Prev)

Cases where a previous CERT/CC advisory has been issued for the same vulnerability account for about 9% of all proprietary software, 0% of open-source software, 9% of the multiple implementations, and 7% of all vulnerabilities that were categorized.

CA-2001-13 (Buffer Over flow In IIS Indexing Service DLL), CA-2001-19 ('Code Red' Worm Exploiting Buffer Over flow in IIS Indexing Service DLL), and CA-2001-23 (Continued Threat of the 'Code Red' Worm) are examples of this. The first advisory (CA-2001-13) identifies a buffer overflow error. The second advisory (CA-2001-19) is about the 'Code Red' worm, which exploited the same buffer overflow error on machines not upgraded after CA-2001-13. The third advisory (CA-2001-23) is an additional alert that indicates that the Code Red worm is still continuing to spread at a rapid pace. Since a fix was provided for the first of these three examples, it is the one attributed with a fix date. Since a new fix was not needed for the other two advisories, they were labeled as having a previously available solution.

Note that, of these example advisories, two were in fact published without waiting the 45 days that is assumed to have passed between the date that the vendor was aware of the vulnerability and the date that the alert was made public. This is due to the urgency of their content, as determined by the CERT/CC. In this case, this was not a problem, since those two advisories were determined to exist previously, and the dates were not considered for them.

Multiple vulnerabilities (MV)

Multiple vulnerabilities are listed within the CERT/CC advisories for 17% of proprietary software, 24% of open-source software, 4% of multiple implementations, and 15% of all alerts studied.

An example of this is CA-2002-09 (Multiple Vulnerabilities in Microsoft IIS), in which there are 10 separate vulnerabilities listed for the Microsoft IIS web server. The vulnerabilities varied from buffer overflows to cross-site scripting vulnerabilities. Fixing these vulnerabilities required downloading and running a cumulative patch that applies 10 separate patches to the IIS server.

No solution available (None).

Vulnerabilities where no solution was found to be available represented 11% of proprietary software, 6% of open-source software, 13% of multiple implementations, and 10% of advisories overall.

An example is CA-2001-03 (VBS/OnTheFly (Anna Kournikova) Malicious Code), which is a VBScript attachment capable of being executed automatically from within Microsoft Outlook. The recommended solution is to install or update current anti-virus software, and to use caution when opening e-mails, both of which do not fix the vulnerable behavior of VBScript within windows.

Another example is CA-1999-17 (Denial-of-Service Tools), where multiple operating systems, including some UNIX, Windows and MacOS, were vulnerable to denial-of-service (DOS) attacks by tools being shared on the Internet. The solution offered was to be aware of the situation, in order to be prepared for a quick response if it happens and to install ingress filtering rules on routers. Neither solution directly changes the vulnerable systems and, as such, is not categorized as solutions provided by the vendors.

Data not available (N/A)

Only one occurrence was found where not enough data was available to provide the fix date. This fell within the proprietary software category, making it 2% of all proprietary software advisories and 1% of all advisories studied. The advisory is provided as the example.

Since CA-2001-16 (Oracle 8i contains buffer over flow in TNS listener) did in fact have a solution, it did not fit under the other scenarios, but it could not fit under the fixed advisories without a date for when the fix was available. Learning more about the vulnerability and the date on which the patch was made available from Oracle required a support contract ID number in order to log in to the Metal ink support site, where the required information was available.

6 Results of analysis

6.1 Consideration of special scenario data

While the special scenarios did not contribute to finding out the amount of days that are required for proprietary or open-source software vendors to provide solutions to vulnerabilities, they did yield some interesting information that could contribute to the results of this paper.

Advisories that were found to be related to previous advisories do, however, appear to provide some meaningful information. Firstly, in this study five were found to exist under proprietary software, where none were found for open-source software. What does this indicate? Since this type of advisory typically referred to the importance (and sometimes urgency) of installing a previously available patch or solution, it could indicate, among other things, that:

(a) administrators for systems containing the proprietary software are less likely to install updates when they are available;

(b) machines with the respective proprietary software are very common or popular in environments where the vulnerability thrives;

(c) vulnerabilities of proprietary software are more likely to be extreme enough to require the release of multiple CERT/CC advisories;

(d) open-source software is not common enough that advisories are worthy of repetition; or

(e) administrators of systems containing open-source software are more likely to keep up to date with software updates that a repeated advisory is never needed.

None of these can be proven without expanding the research to other areas, but the existence of repeated advisories implies that there is a greater problem in either the proprietary software itself or in those responsible for updating the proprietary software in these cases.

More advisories were found to have multiple vulnerabilities for proprietary software than for open-source software. However, since there were more advisories for proprietary software, the percentage of multiple vulnerabilities found in comparison to the other scenarios was greater in open-source software. Some possible implications could be that:

(a) open-source vendors might wait longer to accumulate enough fixes to create a new release (perhaps because, at a given moment, there are fewer resources to apply to the project than a commercial software vendor would have); or

(b) open-source vendors might make the changes available though nightly builds of the packages, so the changes are available but more time passes between major releases of the software than for comparable proprietary products.

The difference between open-source and proprietary software are close enough in this category such that no additional information can be determined.

Advisories where no solution was available can offer some insight. A higher percentage of advisories without a solution fall under proprietary software than under open-source software. All the advisory notes suggested other methods to avoid exploitation of the vulnerabilities. Interestingly, the open-source vulnerability could have been considered to be a previous vulnerability, but CERT/CC assumed that the administrators for the open-source software had already updated the software if they knew it was installed, so they recommended uninstalling the software. All the proprietary software problems suggested installing or updating the current anti-virus software. This shows that the computers with the vulnerability are more likely to be workstations, since they are running e-mail software that can automatically execute a virus or Trojan. It also shows that the proprietary software vulnerabilities in these cases are more easily fixed by using anti-virus software than changing the features of the software that automatically executes the virus.

The 'information not available' (N/A) special scenario is interesting, because the only case that existed was proprietary software. Had more of this type existed, it might have provided significant input, since this categorization implies a restriction of information that would appear to more closely represent proprietary software. There have been times when information has been limited with open-source software, such as vulnerabilities in the OpenSSH software. But, once a solution was made available, all the information, as well as the reasoning for restricting the information, was made public. Since only one case exists in this study, no significant information can be derived from this piece of data.

Lastly, one can argue that the sample sets shown are CERT advisories, thus any conclusions formed apply only to the CERT advisories. While there are certainly other types of computer and network security advisories, it seems that the CERT advisories are the most widely distributed — both by CERT itself and by other security sites. We also considered the CERT Vulnerabilities list, but they varied greatly in severity. Also, there were so many (3222 in the first three quarters of 2002 alone) that it would not have been possible to gather the information for each item if we wanted the data to span more than one year.

6.2 Percentage of vulnerabilities fixed

Before moving on to the amount of time required for vulnerabilities to be fixed, one other piece of data will be compared — the percentage of vulnerabilities fixed. This data can be found in Table 5. Note that the software type classifications for multiple implementations (MI) and non-software (NS) do not have columns in Table 5, but their numbers are still counted in the overall column.

Out of all the advisories that were determined to be based on proprietary software, 61% were found to be fixed. The same figure for open-source software was 71%. These numbers were found by taking the total number of solutions made available for the software type divided by the number of advisories for that type. Contributing to the difference in those figures are the vulnerabilities found to be related to previous advisories and advisories with multiple vulnerabilities. Since those advisories could not be considered to be fixed, they became part of the 'unfixed' group.

By not counting the previously alerted and multiple vulnerability advisories in the total

Table 5: Comparison of Results

	PSS	OSS	Overall
Total Advisories	54	17	96
Total Advisories, excluding Prev and MV	40	13	75
Fixed Advisories	33	12	45
Percentage Fixed	61%	71%	47%
Percentage Fixed, excluding Prev and MV	83%	92%	60%
Average Number of days for Solution	52.45	28.75	46.13
Standard Deviation of Days for Solution	30.25	23.82	30.33
Minimum Days for Solution	31	-33	-33
Maximum Days for Solution	198	45	198

number of advisories for each type of software, the percentage fixed for proprietary software is 83% and for open-source software is 92%. These averages are much higher and seem to better reflect the success of both types of software vendors in fixing vulnerabilities when they are found.

Whether counting or ignoring the multiple and previous vulnerabilities, an approximate 10% difference was found between the number of solutions offered between the two types of software, with open-source having the advantage in both cases. This is a significant difference. While it was not the focus of this research, it is interesting to note that, when looking at CERT/CC advisories, more of the open-source-related advisories have solutions than for proprietary software.

It is true that different types of vulnerabilities require different types of fixes. Some are relatively easy to fix (e.g. a buffer flow), while others may require substantial time and effort to locate them and correct them. A concern that may arise is as follows: there may not be enough detail to know how the above difference contributes to the differences in fixes (and that certain open-source vulnerabilities cannot be fixed). Thus, could the analysis (where unfixable vulnerabilities are excluded) bias the study? We cannot really tell how much, or if at all, it biased the study. Our paper looks at the development approaches to fixing a vulnerability and finds that it seems to make some difference. However, another thing to consider is how the original code was written and if, for example, having more eyes looking at the code during original development might cause it to have fewer 'unfixable' errors, it could in fact be biasing the study to exclude the unfixable ones.

Here are some reasons why we excluded the unfixed/unfixable ones:

> Probably the greatest reason is that our goal was to find out how long it took to fix vulnerabilities. Unfixable vulnerabilities would contribute no time frame to enhance this calculation. So, our approach was to treat every occurrence equally by excluding them all, instead of essentially counting them all as failures, despite the varying amounts of credit used by the development process might have toward the 'fixability' of the vulnerability.
>
> There were more unfixable vulnerabilities for MI (multiple implementation) software than for open-source or proprietary software, and the levels between open-source and proprietary software did not appear to be so different that we could consider it to be a significant piece of data.

6.3 Time required to provide solutions and source code availability

We now look at the main focus of this paper: can either proprietary software or open-source software provide a security advantage in the amount of time that is needed between the date on which a vendor is made aware of a vulnerability and the date on which a solution can be provided? Table 5 shows a comparison of some of the main results discussed.

As explained in section 4.4, a 45 day period has been assumed to have passed between the date on which CERT/CC was made aware of the vulnerability and the date on which the advisory is made public, according to information available on the CERT/CC web site. All calculations for the amount of time required for a solution are based on the date exactly 45 days prior to the public date of the advisory.

For proprietary software, the minimum amount of time required to create a solution was 31 days. The maximum time was 198 days for CA-2001-05. While 198 days was nearly four times the average, advisory CA-2001-15 required 107 days until a solution was available. Since the number of days that was needed to provide solutions varied widely in both directions, no data were labeled as outliers and excluded from the statistics.

For open-source software, the minimum was a negative 33 days for advisory CA-2000-22, which indicates that the solution for that specific vulnerability had been available for 33 days before the assumed vendor notification date. Because the solution was not tied to any previous advisories, it did not fit within the category for advisories based on a previous advisory. The maximum number of days required for a solution to be made available was 45 days, which means that the solution was made available on the date the advisory was made public.

The average number of days required for solutions to be made available was 52.45 days for proprietary software and 28.75 days for open-source software. (Even adjusting the averages by removing the two large values for the proprietary software and the negative value for the open-source software gave averages that still differed by more than 10 days, with open-source software still being the faster of the two.)

By calculating the standard deviations, we see that most proprietary software solutions were made available within 30.25 days of the 52.45 day average. The standard deviation for open-source software was 23.82 days. A lower standard deviation indicates a stronger pattern, or a more solid tendency for similar values to reoccur, and can be seen more in the numbers for the open-source software.

It has been found from this data that open-source software vendors provide solutions to advisories an average of 23.7 days sooner than vendors for proprietary software. This is a significant difference. The standard deviation of 23.82 days suggests that not just the average but a majority of open-source solutions was provided before the average solution was available for proprietary software. This does not suggest that open-source software is necessarily better than proprietary software but that, in environments where it is essential to fix security vulnerabilities as soon as possible, it is very likely that an open-source application would have a solution available sooner.

Some of the proclaimed advantages of open-source software may contribute to this, such as the greater number of pairs of eyes available within the online community to attack a problem when it is discovered, the ability of end-users to make changes to the source code and recompile the application rather than wait for a vendor to create patches or a new release and distribute it, and even the cooperation and joint analysis of the vulnerability within the online community (as demonstrated in the case of the Internet Worm). All these can help to bring about a solution more rapidly. In contrast, the lack of information shared with the general public during the fixing of vulnerabilities in proprietary software may in fact slow down the process.

7 Conclusions

Computer systems connected to networks and to the Internet are never totally secure. When a vulnerability is discovered, the importance of fixing that vulnerability promptly grows as the importance of maintaining the availability of that computer system grows. This study looked at determining if a difference could be found in the amount of time needed to provide a solution for both open-source software and proprietary software. The aim was to provide additional guidance in software selection when both options are available, using security advisory publications from the CERT Coordination Center for the last four years.

The findings were that it took on average about 23 days longer for solutions to be found and made available for proprietary software than for open-source software, and that the majority of open-source solutions had been provided by the average time a proprietary solution had been provided. While many studies seem to differ on whether the advantages given for using open-source software really make it more secure, this study has found that some of those advantages may in fact contribute to quicker solutions when security vulnerabilities are found.

7.1 Challenges

In this section, we discuss challenges to collecting the data for this paper that may have affected the quality or accuracy of the data.

Determining the date on which a vendor was notified

As explained in section 4.4, determining the date on which vendors were notified of vulnerabilities in their product is nearly impossible in most cases for proprietary software. Furthermore, this information does not appear to be readily available from CERT/CC. E-mails sent to CERT/CC requesting information about when vendors are notified was automatically replied to with an informational message, but never followed up with a personal reply.

Determining software types

While the mention of certain software companies or the name Linux makes determining the software type easier, software applications were found in CERT/CC advisories that needed to be researched to determine their type. In cases where an application or library was used in a UNIX environment, the assumption that the product was open-source could not be made. Several versions of UNIX are in fact proprietary software, as well as much of the software that is used on them.

Understanding the vulnerability and its solution

In order to know if a solution existed, and when a solution was available that prevented further exploitation of the vulnerability, certain information had to be known. This information ranged from understanding the service or type of application that was vulnerable, to determining if a previous solution to the vulnerability existed. Anything that was not known needed to be researched, mostly using online sources.

7.2 Future work

Locate an official resource for the date a vulnerability was discovered

Since an accurate calculation of the amount of time needed to fix a vulnerability is based on having correct dates, it would have been helpful to have a trusted resource with those dates. Perhaps CERT records the original dates when they are made aware of vulnerabilities. Ideally, CERT/CC could provide this information after a solution has been provided, and contact the vendors when the date is known. This could solve inaccuracies in the data and potentially provide more accurate results.

Use CERT/CC vulnerabilities instead of advisories

The CERT/CC vulnerability database has far more information than the list of advisories, so much more data could be collected while applying the same analysis. More data could identify trends more accurately. However, with CERT/CC advisories there is a built-in equalizer, in that each one is selected because it has reached a certain level of urgency. Not all CERT/CC vulnerabilities are similar in their level of urgency, although the overall conclusions could possibly benefit from an increased number of statistics. It would be important to investigate the potential benefit thoroughly before undertaking this task, since the total number of vulnerabilities in recent years are in the thousands.

Determine the contributing factors for more rapid solution

It is likely that some of the attributes of the open-source development process may contribute to having solutions that are generally available more quickly than for proprietary software. It would be beneficial to determine which attributes are more significant, and raise awareness of them within the software community. It would also be interesting to look for ways to apply some of the approaches to commercial products, either by releasing some of the proprietary information when the benefits outweigh the needs, or finding a work-around to divulging source code or proprietary information.

References

[1] Littlewood, B. and Strigini, L., 2000. Software reliability and dependability: a roadmap', *Proceedings of the International Conference On The Future of Software Engineering (Limerick, Ireland)*, pp. 175-188.

[2] Cowan, C., Pu, C. and Hinton, H., 1998. Death, taxes, and imperfect software: surviving the inevitable, *Proceedings of the 1998 Workshop on New Security Paradigms (Charlottesville, VI, USA)*, pp. 54-70.

[3] Neumann, P.G., 1995. Computer Vulnerabilities: Exploitation of Avoidance, *Communications of the ACM,* Vol. 38, No. 6, p. 138.

[4] Devanbu, P.T. and Stubblebine, S., 2000. Software engineering for security: A roadmap, *Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland)*, pp. 227-239.

[5] Garfinkel, S. and Spafford, G., 2003. *Practical UNIX & Internet Security*, 3rd edn (O'Reilly & Associates).

[6] Pethia, R., Paller, A. and Spafford, G., 2000. Consensus Roadmap for Defeating Distributed Denial of Service Attacks. Members of CERT/CC at Carnegie Mellon University, The SANS Institute, and The Center for Education & Research in Information Assurance & Security (CERIAS) at Purdue University, respectively. http://www.sans.org/ddos`roadmap.htm.

[7] Neumann, P.G., 2000. Robust Non-proprietary Software, *Proceedings of the 2000 Symposium on Security and Privacy* (IEEE Computer Society, Oakland, CA, USA), pp. 122-123.

[8] Martin, W. B., White, P. D. and Van eet, W. M., 2000. Government, industry, and academia: Teaming to design high confidence information security applications, *Proceedings of the Third Workshop on Formal Methods in Software Practice (Portland, OR, USA)*, pp. 37-47.

[9] Neumann, P.G., 1998. Inside Risks: Robust Open-Source Software, *Communications of the ACM*, Vol. 41, No. 2, p. 128.

[10] Stallman, R. M., 1994. *Why Software Should Not Have Owners* http://www.gnu.org/philosophy/why-free.html.

[11] Rochilis, J. A. and Eichin, M. W., 1989. With microscope and tweezers: The worm from MIT's perspective, *Communications of the ACM*, Vol. 32, No. 6, pp. 689-698.

[12] Spafford, E. H., 1989. Crisis and aftermath, *Communications of the ACM,* Vol. 32, No. 6, pp. 678-687.

[13] CERT/CC staff, CERT/CC Statistics 1988-2002, last updated 4 October 2002, http://www.cert.org/stats.

[14] CERT/CC staff, CERT/CC Vulnerability Disclosure Policy, 2000, http://www.kb.cert.org/vuls/html/disclosure.