

Protection or Sabotage?

An Ethical Analysis of Software Vulnerability Disclosure

By William Whitney

June 1, 2009

CSC 300

Abstract

In recent decades, the internet has grown dramatically, connecting computers across the world. As a result of this interconnected global network software is now more vulnerable to attack than ever. Many of these software attacks are well known exploits on commercially available software. This paper uses the Witty Worm exploit as a case study to explore the ethics of fully disclosure relating to software vulnerabilities. Analysis will be performed using utilitarian principles, deontological reasoning and the software engineering code of ethics. The primary arguments examined will relate to user's right to protection versus the benefits of non-disclosure. The conclusion is that disclosure policy adopted by Internet Security Systems (ISS) was ethical however; it did slightly raise the public's risk of being exploited.

Table of Contents

Introduction.....	1
Known Facts.....	1
Research Question.....	3
Importance	3
Arguments - Not Ethical.....	4
Security Advisory Provided Necessary Information Needed to Create Exploit	4
ISS Issued Security Advisory Before System Administrators Had Time to Apply Patch	4
Initial Release of Vulnerability Patch May Have Led to Exploit	4
Arguments – Ethical.....	5
Conception of Witty Worm May Have Occurred Before ISS Vulnerability Disclosure.....	5
Witty Worms Creator May Have Had Inside Knowledge of ISS Products	5
Analysis.....	6
Introduction.....	6
Utilitarian Analysis	6
Introduction.....	6
If a flaw is already known by the bad guys then the good guys should also know to protect themselves	6
Security vulnerabilities should be kept secret because they increase the risk of attack	7
If a security vulnerability report is published early then administrators can get a head start on locking down vulnerable machines	8
When security vulnerabilities are disclosed users ultimately become safer.....	9
Utilitarian Conclusion	10
Deontological Analysis.....	11
Software vulnerabilities should be published in a general way to obscure technical details of the attack.....	11
Software vulnerabilities should be fully disclosed after the vender has been notified and a fixed amount of time has been given for a patch to be created.....	12
Software vulnerabilities should be disclosed so that users can make an educated decision about purchasing or using a particular piece of software.....	13
Deontological Conclusion	14

Software Engineering Code of Ethics.....	16
If a software vendor knows of a security exploit then they should inform the public.	16
Vendors should keep quiet on security issues and integrate fixes quietly into the next version of their product.....	17
Software Engineers Should Disclose Technical Details of Software Vulnerabilities.	18
Vendors Must Make a Patch if a Software Vulnerability is Reported to Them.	19
Vendors Must Inform Users When Their Software is Actively Being Exploited.....	20
Vendors Should Record and Report Vulnerability Statistics Involving Their Product.....	21
Software Engineering Code of Ethics Conclusion	22
Conclusion	23
Works Cited	25

Introduction

In recent years, software security has experienced an increase in visibility. Unfortunately, much of this attention is due worms and viruses [12]. Even worse, many of these attacks exploit systems using known security vulnerabilities. The recent “Conficker” worm for example, exploits a known security vulnerability for which a patch exists [1]. This begs the question how should security vulnerabilities be disclosed and under what conditions? This paper will examine the ethical basis of fully disclosing software vulnerabilities using “Witty Worm” as a case study. The ethics of vulnerability disclosure will be analyzed first using utilitarian principles, second through the application of deontological reasoning and finally the software engineering code of ethics. The ultimate goal of the analysis section is to conclude that disclosing software vulnerabilities is in fact ethical.

Known Facts

March 8, 2004 eEye Security discovers a flaw in network security products BlackICE and RealSecure made by Internet Security Systems (ISS) [2]. This flaw exploits the protocol analysis module (PAM) allowing malicious ICQ packets to create a stack overflow [2]. This overflow was particularly troublesome because the return address on the stack was overwritten potentially allowing unauthorized code to be executed [2].

March 9, 2004 a patch was released by ISS that fixed vulnerabilities in BlackICE and RealSecure found by eEye [6]. ISS also recommends on their website that companies and users block source port 4000 as a proactive measure against potential worm traffic [6].

March 18, 2004 ISS releases a vulnerability advisory for its protocol analysis module (PAM) which is used by BlackICE and RealSecure software [3]. This advisory summarizes the insufficient size checks on certain protocol fields that are used in ICQ response packets [3]. ISS reports that insufficient boundary checks might make it possible for an attacker to cause memory corruption with the potential of remote exploitation [3].

March 20, 2004 ISS releases a security alert announcing that unpatched versions of BlackICE PC Protection are spreading a worm based on the security advisory released March 18, 2004 [4]. This worm dubbed “Witty Worm” and takes advantage of a stack-based overflow in the PAM ICQ protocol-parsing tool [4].

Once Witty Worm was released, it spread very quickly infecting nearly all vulnerable hosts within 45 minutes [5]. Witty Worm worked first by sending out 20,000 packets to random IP addresses, second deleting a random block from the hard drive and finally repeating the process until the computer became inoperable [7]. At Witty Worm’s peak over 90Gbits/Second worth of packets were being flooded on to the internet and 12,000 computers were infected [5].

Many for the following reasons view witty Worm as unique. First, it was extremely successful by infecting nearly all vulnerable hosts within 45 minutes [8]. Second, it was speedily written and released less than three days after the initial vulnerability report by ISS [8]. Third, it was written bug free and delivered only a single randomly sized packet [8]. Finally, it was deadly to its hosts because it randomly deleted sections of the hard drive [8].

Research Question

Was it ethical for Internet Security Systems (ISS) to disclose the software vulnerabilities found in BlackICE and RealSecure that were ultimately exploited by Witty Worm less than three days later?

Importance

Concluding the ethicality of the vulnerability disclosure that ultimately led to Witty Worm is important for a couple of reasons. First, software engineers face a unique problem when it comes to product testing because expected output for tests often does not exist or is impractical to determine [31]. As a result, software engineers are never fully able to verify their code is completely correct. Secondly, today's economy utilizes technology such as cell phones, e-mail, medical equipment, and computers more than ever [32]. If software engineers are never able to prove their code is correct and reliance of technology is increasing then the only inevitable result is software vulnerabilities. The unknown as of yet is how to handle and disclose these vulnerabilities as they arise. Handling software vulnerabilities ethically should be a primary interest to software engineers because potential danger exists for the customer, employer and the public [40]. While this paper only handles the ethicality of the disclosure, utilize by Witty Worm perhaps one day this case study can be used to determine the optimal solution for vulnerability disclosure at large.

Arguments - Not Ethical

Security Advisory Provided Necessary Information Needed to Create Exploit

Some argue that when ISS released their security advisory on March 18, 2004 they provided the Witty Worm attacker all the necessary information needed to make a viable exploit [2]. This argument is based on the fact that the March 18, 2009 advisory detailed the faulty module and protocol that could be used to create a stack overflows in the BlackICE and RealSecure software [3]. Individuals on this side of the argument believe that details regarding the nature of this threat ultimately led to the implemented Witty Worm attack.

ISS Issued Security Advisory Before System Administrators Had Time to Apply Patch

Another argument against ISS's release of the March 18, 2004 security advisory is that not enough time passed to allow administrators to install the patch [13]. Administrators only had nine days to download the patch while it was still concealed. Installing patches for this type of software would require a system administrator. This is especially true for RealSecure since it is a software firewall running on a dedicated server.

Initial Release of Vulnerability Patch May Have Led to Exploit

A third view is that the initial patch put out on March 9, 2004 was disassembled to reveal the flaw that led to Witty Worm's creation [4]. Individuals in this camp argue that merely publishing a patch to a system provides a risk. Hackers may reverse engineer a patch in order to find the security vulnerability that was fixed [15].

Arguments – Ethical

Conception of Witty Worm May Have Occurred Before ISS Vulnerability Disclosure

Some believe that Witty Worm may have been implemented and tested long before its nearly immediate appearance [2]. This view assumes that the ISS security advisory was just the opportunity that Witty Worm needed to thrive. Based on the sophistication of the worm it is hard to argue that Witty Worm was assembled in just a few days. First, it was implemented using only a single UDP packet meaning that it could travel across almost any network segment [2]. Second, it avoided many common worm pitfalls [2]. Finally, it was written bug free [8]. As a result of these facts some believe that Witty Worm was delivered quickly because the worm was already built thus the ISS disclosure only partially led to its creation.

Witty Worms Creator May Have Had Inside Knowledge of ISS Products

Another view on Witty Worms creation is that its author had inside knowledge about ISS products [2]. This view stems from the fact that one hundred and ten hosts were used to seed the initial network [7]. Even at Witty Worms height only twelve thousand hosts were infected so where did the first one hundred and ten come from? If Witty Worm's creator did have inside knowledge then chances are the disclosure of the vulnerability had little effect.

Analysis

Introduction

The following sections will examine the ethical basis of a wide variety of arguments pertaining to the vulnerability disclosure that led to Witty Worms creation. Analysis will occur using the ethical principles of utilitarianism, deontology and the software engineering code of ethics. The final goal is to determine if the disclosure method utilized by ISS was in fact ethical.

Utilitarian Analysis

Introduction

This section will utilize the ethical theory of utilitarianism to show that ISS's release of the security advisory that ultimately led to the Witty Worm exploit was in fact ethical.

Utilitarian analysis revolves around creating the greatest good for the greatest number [9].

If a flaw is already known by the bad guys then the good guys should also know to protect themselves [10].

This argument rests on the fact that many communication networks exists for hackers [14]. Once a software vulnerability is discovered and released to the hacker community it is very likely that an exploit will soon be distributed. In fact, Witty Worm followed this same pattern. Less than three days after the initial ISS security advisory the worm was released [8].

Not openly distributing information about vulnerabilities after they have been distributed within the hacker community is unethical from a utilitarian view of maximizing

happiness. If only the hacker community has information regarding an exploit then only the hacker's happiness can be increased. Hackers make up only a small subset of the entire user population thus overall happiness is reduced. One way to equalize happiness is to share the information regarding the vulnerability with the public. In this case little harm is done by sharing the information because the hackers already know about the vulnerability. Thus under utilitarian analysis it is ethical to fully disclose security vulnerabilities in the case that the hackers already know about it.

Security vulnerabilities should be kept secret because they increase the risk of attack [10].

Proponents of this view argue that most of the risk in disclosing security vulnerability lies in the disclosure itself [16]. By disclosing the nature of a security vulnerability hackers now know a problem exists, roughly what it looks like and finally where to find it. The burden of finding software vulnerabilities at this point is no longer solely the responsibility of the hacker community. It is argued that all the hacker needs to do now is create and deliver the exploit using the security advisory as a guide [2]. Even worse many exploits are streamlined through the process of scripting [16]. Once an exploit has been scripted it can be utilized by almost any attacker even those with little or no skill [16].

From a utilitarian viewpoint, it does make sense to keep security vulnerabilities secret for at least a small amount of time. If a software vulnerability is reported immediately overall happiness will be reduced because hackers will start building an exploit at the same time the software manufacture is working on a patch. In the period before a vendor issues a patch users have few options for protection. First, users might simply do nothing and hope they are not

attacked, second they might stop using the product completely and finally they may implement some sort of temporary work around if it exists [17]. None of these three options does anything to increase the happiness of the user. Yet if the vendor is able to keep the security vulnerability secret for long enough to develop a patch and start its distribution overall happiness is increased. Vendor patches protect against exploit and allow the software to be utilized [17]. Since the idea of full and immediate disclosure is not completely ethical from a utilitarian viewpoint and keeping software vulnerabilities completely secret limits happiness the middle ground between these two extremes is the most ethical because it allows vendors time to create a patch yet still distributes information in a timely fashion.

If a security vulnerability report is published early then administrators can get a head start on locking down vulnerable machines [10].

This argument aims to place hackers and users on a level playing field regardless of whether or not a patch currently exists. Under this model, users and hackers are given the same amount of time to secure or exploit the system. Currently a mailing list called “BugTraq” operated by SecurityFocus.com serves as the cornerstone for the full disclosure moment [19]. This organization’s mailing list publishes full details of security vulnerabilities immediately [19].

From a utilitarian viewpoint immediately, publishing software vulnerabilities under the heading of full disclosure does not increase overall happiness. If the information is released immediately, the vendor and the hackers are in a competition to develop a fix or an exploit. This competition might be ethical since vendors should not have coded an exploit into their system but unfortunately, the vendor and the hackers are not the only ones involved. In both

cases the either the development of an exploit or a patch a one too many relationship exists. The vendor only has to issue one patch to protect many systems likewise a devious hacker needs just one exploit to cause untold harm. Thus, there exists a substantial risk that net happiness will be reduced by one hacker with knowledge of a software vulnerability. What about the happiness of the system administrator who wants to proactively secure his or her system? It is true that not disclosing the security vulnerability immediately reduces their happiness but it is hard to justify their happiness at the potential expense of many. Yet interestingly even when a patch is available many users and system administrators do not install it immediately [20]. If patches are not applied quickly then perhaps it shows that little happiness is derived from secure systems. Thus it is unethical immediately distribute the details of a software vulnerability because it is not worth putting the happiness of a majority of users at risk for a limited amount of system administrators who may or may not install the patch in a timely manner.

When security vulnerabilities are disclosed users ultimately become safer.

Proponents of this argument claim that by fully disclosing software vulnerabilities everyone in the software community benefits. A published vulnerability does create more exploits but also raises public awareness regarding the vulnerability type [21]. It is argued that by raising public awareness system security will be increased because other developers and software vendors will know what software engineering methods are particularly dangerous.

According to utilitarian principles, this argument does in fact maximize overall happiness. If security vulnerabilities are disclosed then developers and software vendors will

know what methods their attackers are currently utilizing. When a vulnerability is disclosed it benefits users beyond those affected by the vulnerable piece of software. This argument can be made since certain types of security vulnerabilities are likely to show up in other applications [21]. Take for instance a buffer-overflow exploit. If many vulnerabilities are being discovered utilizing this attack avenue then developers are going to give special attention to this issue. By eliminating popular attack avenues hackers will be forced to find more exotic and novel ways to exploit software [18]. The hope is that more exotic and novel attacks are less likely to affect the average user. Thus, overall happiness is increased since vendors and developers are able to make more secure software by knowing where attackers are likely to hit.

Utilitarian Conclusion

From a utilitarian perspective, the process ISS followed to disclose the vulnerability later utilized by Witty Worm was ethical. ISS issued a patch within one day of being made aware of the vulnerability by eEye security. At this point, the public was still unaware of the vulnerability and a patch was produced faster than most system administrators could have secured their systems. ISS also waited nine days to disclose information about the vulnerability. This allowed proactive users and system administrator's time to install the patch. The advisory released nine days later served as a notification to remaining users. It is unfortunate that Witty Worm exploit followed so quickly after the release of the advisory. Yet net happiness was still increased, after keeping the details of the vulnerability a secret for nine days the risk of secrecy grew. As time goes on information regarding this vulnerability could have slipped into a communication network used by hackers or perhaps the patch may have even been reverse engineered itself.

Given that ISS's disclosure process sidestepped the potential pitfalls of disclosing too early and not providing a patch in a reasonable time period the ISS disclosure policy was in fact ethical.

Deontological Analysis

Introduction

The following section aims to prove that ISS's disclosure of the vulnerability that later led to Witty Worms creation was in fact ethical using deontological reasoning. Deontological reasoning is based on following a set of moral rules rather than judging the "goodness" of the resulting action [23].

Software vulnerabilities should be published in a general way to obscure technical details of the attack [11].

Proponents of this view argue a detailed security advisory provides a road map for hackers to follow when creating an exploit [2]. If the technical details are obscured enough then users will know that the software they are using has a vulnerability but the hackers will not know how to quickly exploit it. The goal of this argument is to meet the responsibility vendors have in informing users of potential danger and the responsibility to protect users from unreasonable harm [22].

From a deontological viewpoint, publishing vulnerabilities in a general way is not ethical. When information is published in a limited fashion, its usefulness becomes diminished. In the case of software vulnerability reports, just saying a piece of software is vulnerable may just stir up fear about an attack rather than provide a way to reduce it [24]. According to Kant moral laws should apply in a manner that everyone can partake of the same action at once without

harm [25]. If all vendors publish security vulnerability advisories without providing any details about how or what is defective then much fear will result from knowing that vulnerabilities exist. Kant also argues that the highest form of good will is to perform ones duty [25]. Vendors should protect their users from danger and harm that might come from the use of their product. When security advisories are published in only a general way, little can be done as a user to protect oneself [24]. Ultimately, under deontological reasoning publishing the full details of a security vulnerability is preferable because it reduces fear and allows users to protect themselves better.

Software vulnerabilities should be fully disclosed after the vender has been notified and a fixed amount of time has been given for a patch to be created [11].

This argument rests on balancing the right of the user to be protected with their right to know if the software they are using is dangerous [22]. When a software vulnerability advisory is released an attack almost always follows [26]. Yet the if the vendor can make a patch while the vulnerability is still under wraps then at least users will be able to protect themselves once the advisory report is released. Unfortunately, it has also been recognized that some vendors are not responsive to security vulnerabilities and only will make a patch when a vulnerability disclosure is threatened [27]. The ultimate goal of this argument is give vendors a chance to make a patch before the hacker community is aware but still keep pressure on the vendor by setting a time table for disclosure.

This argument is ethical from a deontological perspective because it has the potential to become a universal rule. Kant argued that moral rules must be optimal such that everyone in

society can perform them at once without harm [25]. When software vulnerability advisories are published before a patch exists it leaves the users open to attack [26]. Yet if vulnerabilities are never disclosed then users are unaware that they are using dangerous products [28]. When vendors are give information about a software vulnerability along with a time table for disclosure a more optimal balance is found. The vendor now has time to create a viable patch and start its distribution to the user base without placing users in undo harm. In addition, the vendor is under a little bit of pressure to complete the patch. In fact, the computer emergency readiness team (CERT) for Carnegie Mellon University currently supports this mode of disclosure [29]. This institution collects software vulnerability reports; notify the vendor and then the public after a fixed timetable [29]. Keeping software vulnerabilities a secret while vendors make a patch on a fixed time table is ethical from a deontological standpoint because no one group is overly taken advantage of when this rule is applied to society at large.

Software vulnerabilities should be disclosed so that users can make an educated decision about purchasing or using a particular piece of software [28].

This view holds that security vulnerabilities are important quality marker [28]. If software vulnerabilities are publicly available then users will be able to make a more educated choice between similar products if security is a high priority for them [28]. This argument aims to install a balance between vendors who rush to market to beat a competitor at the expense of security [30].

From a deontological perspective, disclosing software vulnerabilities is ethical. Kant suggests that we should use people as an ends rather than a means [25]. This means that vendors should not hide information about vulnerabilities simply because they can. If a vendor

has a mistake in their product then they should take responsibility to ensure users know the risks and benefits of their software. Much of what happens inside a software system is obscured from the user. Tens of thousands lines of code just become just a single installer file hiding the inner workings of the system. Without software vulnerability disclosures users might be led into a false sense of security believing that their product is completely safe. Thus from a deontological perspective of treating the customers as ends rather than means software vulnerabilities must be disclosed.

Deontological Conclusion

In the case of Witty Worm ISS side stepped many deontological ethical dilemmas. First, ISS resisted publishing the details of the security vulnerability that led to Witty Worm in a general way. The March 18, 2004 security vulnerability detailed exactly what software was vulnerable and which modules contained the potential exploit. From a deontological perspective the users of ISS software were treated as ends by giving them all the necessary information to protect themselves rather than abstracting away the vulnerability information. Second, disclosure only occurred after ISS created a patch for the affected products BlackICE and RealSecure. In fact, the patch existed since March 9, 2004 meaning that a time window existed for users to protect themselves without worrying about harm from hackers. Finally, users of BlackICE and RealSecure were given a necessary piece of information they could use to determine if they wanted to continue using these products. Knowing if their products were vulnerable to attack is likely important to BlackICE and RealSecure users because these products are security products themselves, software firewalls. Ultimately based on these actions ISS was ethical in its disclosure of the software vulnerability that led to Witty Worms

creation.

Software Engineering Code of Ethics

Introduction

The following section will utilize the Software Engineering Code of Ethics to establish whether the security advisory released by ISS that led to Witty Worm was an ethical disclosure. The software engineering code of ethics is “jointly approved by the ACM and the IEEE-CS as the standard for teaching and practicing software engineering” [22]. This code of ethics ultimately aims to help software engineering professionals responsibly manage the great responsibility entrusted to them as a part of the computing profession [22].

If a software vendor knows of a security exploit then they should inform the public.

This argument is based on the notion that software vendors have a responsibility to inform the public of dangers pertaining to the use of their product. When the vendor withholds vulnerability information, users may be blindsided by an attack [28]. This is especially true if the software vendor knows exploit information is being distributed within the hacker community.

Section 1.04 of the software engineering code of ethics states that software engineering professionals should disclose any actual or potential danger that might arise from use of the product [22]. Section 6.07 requires that software professionals be accurate in stating the characteristics of software for which they work [22]. These sections make a strong case for software engineers to disclose software vulnerabilities found in their product for two reasons. First, keeping software vulnerabilities secret does not make users aware of the potential

dangers that may result from the use of the software. Second, limiting disclosure does not accurately describe the characteristics of the software to the user. Thus, it is ethically required that vendors disclose to the public any software vulnerabilities found.

Vendors should keep quiet on security issues and integrate fixes quietly into the next version of their product.

This argument rests on the fact that publishing information about security vulnerabilities often leads to an initial exploit [26]. After the initial exploit has been found it is often scripted which allows the attack to be carried out by hackers with little or no skill [26]. Unfortunately, this issue is not exactly clear cut because tools also exist that allow hackers to reverse engineer patches in order to determine the fixed vulnerabilities [15]. Once hackers know what portion of the code was patched they can target this section of code in unpached versions of the product.

Section 5.01 of the software engineering code of ethics states management should invoke effective procedures for increasing quality and reducing risks [22]. Section 4.01 advocates tempering all technical judgments with views that maintain human values [22]. These two sections of the software engineering code strongly contradict integrating fixes quietly into new versions of the product for the following reasons. First, when security vulnerabilities are kept secret and integrated into iterative versions risk is not reduced because vendors must actually release iterative versions of the product. Second quality is not increased because now the software development team has to worry about the addition of new features and the creation of a patch. Finally, human values are not advocated because the user remains

at risk while a version upgrade is being made. Thus integrating secret vulnerability fixes into iterative versions of the product is unethical under the software engineering code of ethics.

Software Engineers Should Disclose Technical Details of Software Vulnerabilities.

This argument rests on the fact that awareness about common exploit avenues will enable software engineers even those on different projects to anticipate common attacks [34]. If many attacks are currently taking advantage of a buffer overflow weakness for instance then developers will all benefit from the knowledge that such attacks are occurring [34]. The PHP scripting language for instance is particularly vulnerable to mysql injection attacks [35]. These attacks allow users to inject malicious mysql like “delete FROM users” [35]. Yet because these attacks are so prevalent, PHP developers are extremely aware of common PHP vulnerabilities [36]. In fact, many websites and journal articles have been written detailing how to detect, perform and patch the most common vulnerabilities [36].

Strong support for disclosure of vulnerability information exists in the software engineering code of ethics. Section 7.02 states that software engineers should assist colleagues in professional development and section 6.03 says software engineers should share knowledge through participation in professional organizations and publications [22]. Imagine for an instance that a popular software toolkit existed for online banking and all banks kept secret a common vulnerability. The safety of the tool kit would suffer because the original software vendor would not know of the vulnerability and new banks using the toolkit would find out about the vulnerability only when they are exploited. Banks withholding this information would be in direct contradiction of section 6.03, which requires software engineering to share knowledge.

This bank also would be in contradiction of section 7.02 also since the software engineer at the bank should assist in the professional development of the software engineer writing the toolkit for online banking. Since software engineers are required to share knowledge and promote the development of the profession, it is unethical to hide the technical details of a software vulnerability. Thus, software engineers must disclose the technical details of software vulnerabilities found.

Vendors Must Make a Patch if a Software Vulnerability is Reported to Them.

This argument rests on the tradeoffs software engineering firms must make between the interests of the firm, employee, customer and public. Software vulnerabilities are defects in the product. Repairing defects cost not only the software firm but also the customer and the public. Starting first with the employer their cost is that of paying the employee to fix the defect, which according to McConnell is 50 to 200 times that of fixing the defect early on [33]. The customer may also pay for the defect by spending time installing the patch, fixing an exploited system, or perhaps being a victim of identity theft. The federal trade commission found victims of “New Account Fraud” a type of ID theft spend on average \$1200 repairing their finances [34]. Finally, the public is affected if the exploit is particularly malicious and does great harm or indirectly through the loss of economic productivity in society.

Section 6.08 of the software engineering code of ethics states, “take responsibility for detecting, correcting, and reporting errors” interestingly it does not say “take full responsibility” which has a unique contrast with section 1.01 accept full responsibility for your own work [22]. In the software engineering code, there is a subtle distinction between

accepting responsibility and taking responsibility. According to section, 1.01 software engineers should always admit mistakes but they are not always bound to fix them? This slight ambiguity ultimately means that software engineers must make tradeoffs as stated in section 1.02 moderate the interests of the software engineer, employer, client, user and public good [22]. If a software engineering firm finds out about low risk security vulnerability in a little used product it would be acceptable for them not to repair the vulnerability as long as it is reported. This is ethically correct under the code because the employer will spend far more resources than the protection the user will gain from it. Yet if the vulnerability were to heavily effect the public and client then the software engineering firm must correct the defect because at the interests of the client and public are higher. Thus, it is not ethically required in every case for software engineers to patch defects yet they must always report them.

Vendors Must Inform Users When Their Software is Actively Being Exploited

This argument rests on the fact that hackers are often attracted to new and novel exploits [37]. Unfortunately, new attacks often spread quickly through the process of scripting [16]. Once the attack is scripted, hackers with little or no skill can easily deploy it [16]. As a result of the scripted attack and the desire of hackers to use new and novel attacks the rate of exploit increases on new vulnerabilities.

The software engineering code of ethics section 1.05 states that software engineers should cooperate in efforts to address matters of grave public concern [22]. Section 2.07 states that software engineers should report significant issues of social concern they know of to their employer or client [22]. These two sections make a strong case that software engineers must

tell users when their software is being actively exploited for the following reasons. First, if the software vendor never admits an exploit is active, they will be unable to cooperate contradicting section 1.05. Second, if they fail to inform the client, they are acting unethically because many users store sensitive information on their computers like financial information [38]. If this information was, easily accessible because of vulnerability it would be a social concern since the credit and reputation of thousands is at risk. Thus, it ethically required that software engineers inform users when their software is actively being exploited because without disclosure there is no way to cooperate on matters of social concern.

Vendors Should Record and Report Vulnerability Statistics Involving Their Product.

This argument is raised by the idea that software vulnerability reports can help consumers determine the quality of their product [39]. Software quality is inherently difficult to determine since the code is usually hidden by one executable file. If one software vendor rushes to market while another delays the vulnerability statistics would provide a measure to determine quality.

Section 6.07 says that software engineers should accurately state the characteristic of the software for which they work and avoid claims that might be speculative, vacuous, deceptive, misleading, or doubtful [22]. Section 3.01 states software engineers should ensure significant tradeoffs are available for consideration by the user and the public [22]. These two sections make a strong case for recording and reporting vulnerability statistics for the following reasons. First, the only way a software engineer could fully avoid speculative or deceptive statements would be to disclose accurate information regarding their products. This is

especially true in the case that the software is supposed to perform a safety or mission critical job. Second, software engineers have a responsibility as stated in section 3.01 to ensure significant tradeoffs are available for consideration. The primary indicator of quality in software is a lack of defects. The existence of defects in a software package is important because according to McConnell the cost of correcting defects cost 50 to 200 times more at later stages in the product lifecycle [33]. If many vulnerabilities are found users may want to evaluate the usage of the product since the cost to fix the vulnerabilities will become large. If the cost becomes too large, the company will either start limiting the releases of fixes or simply go out of business. The only way users will have enough information to make this tradeoff for themselves is if vendors disclose vulnerability statistics. Thus, it is ethically required by the software engineering code of ethics that vendor's record and report vulnerability statistics regarding their product in order that misleading statements are avoided and important tradeoffs are available for consideration.

Software Engineering Code of Ethics Conclusion

ISS's release of the advisory that was later exploited by Witty Worm fits well within the ethical guidelines set out by the software engineering code of ethics. First, ISS knew about a vulnerability in their product patched it and released the vulnerability information to the public. As a result of these actions they met their obligation to disclose any potential risk users may face as a result of this product. Second, ISS did not fall victim to the theory of bundling fixes with version upgrades. As a result of this action users were able to patch their system immediately and protect themselves. Third, ISS released the technical details of the

vulnerability as required by the software engineering code in order to further the development of the profession. Forth, the decision to create a patch correctly considered the interests of the public, employer and customer as required by the software engineering code since exploits involving memory overflow are particularly harmful and can allow the return address on the stack to be overridden. Fifth, RealSecure and BlackICE users were notified promptly when Witty Worm attacked their products, which is consistent with the codes requirement to report incidents of social concern. Finally, ISS make this vulnerability information available on their website complying with the software engineering code's requirement that users have adequate information regarding tradeoffs. Due to this analysis, it can be concluded that ISS acted ethically in its disclosure of the security vulnerability that ultimately was utilized by Witty Worm.

Conclusion

ISS's security disclosure that ultimately became the catalyst for Witty Worms creation has now been analyzed using utilitarian analysis, deontological reasoning and the software engineering code of ethics. The utilitarian analysis concluded that ISS took actions that ultimately increased overall happiness. Primarily ISS worked in a quick and straightforward manner to patch and disclose the vulnerability. As a result of these actions many potential opportunities to reduce net happiness were avoided such as the hackers only knowing about the exploit, limited details existing about how to secure your system and finally the attacker's methods were not kept secret increasing overall awareness of this method of attack. The deontological analysis concluded that ISS did follow morale rules that if applied to society at large would be beneficial. First, ISS published the vulnerability completely providing enough

detail such that concerned users and administrators could act on the information, second the patch for the vulnerability was available nine days before the actual disclosure of the vulnerability. Finally, by disclosing the details of the vulnerability ISS customers could judge the quality of the application and make an informed decision about continuing its use. The software engineering code of ethics analysis concluded that ISS acted responsibly as a professional software engineer. First, ISS knew about the vulnerability, patched it and then disclosed the information fulfilling its responsibility to disclose potential dangers that may result from the software. Second, ISS resisted the temptation to integrate the vulnerability fix into an iterative version of their product, which complies with their responsibility to take actions, which increase quality and reduce risks. Third, the information regarding the defect was disclosed allowing ISS to foster a potential advance in the software engineering profession by making certain dangers and common hackers exploits available to the industry at large. ISS's actions are ethical according to utilitarian analysis, deontological reasoning and the software engineering code of ethics. Ultimately leading this paper to conclude that the disclosure policy undertaken by ISS was ethical itself.

Works Cited

- [1] Larkin, Erik. Protecting Against the Rampant Conficker Worm. 16 Jan 2009. PCWorld 3 May 2009.
<http://www.pcworld.com/article/157876/protecting_against_the_rampant_conficker_worm.html>.
- [2] Weaver, Nicholas. Reflections on Witty: Analyzing the Attacker. June 2004, USENIX, 3 May 2009.
<<http://www.sagecertification.org/publications/login/2004-06/pdfs/weaver.pdf>>.
- [3] Vulnerability in ICQ Parsing in ISS Products, 18 March 2004. IBM Internet Security Systems. 28 April 2009 <<http://www.iss.net/threats/166.html>>.
- [4] BlackICE Witty Worm Propagation, 20 March 2004. IBM Internet Security Systems. 28 April 2009
<<http://www.iss.net/threats/167.html>>.
- [5] The Spread of the Witty Worm, 18 Nov 2008. CAIDA. 3 May 2009
<<http://www.caida.org/research/security/witty/>>
- [6] Lyman, Jay. Witty.A Worm Targets BlackICE Firewall Vulnerability, 22 March 2004. TechNewWorld.
3 May 2009 <<http://www.technewsworld.com/story/33199.html?wlc=1241383398>>
- [7] Shannon, Colleen. The Spread of the Witty Worm. July/Aug 2004. IEEE Security & Privacy 3 May 2009
< <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.3096&rep=rep1&type=pdf>>
- [8] Schneier, Bruce. The Witty worm: A new chapter in malware. 2 June 2004. Computerworld 3 May 2009.<<http://www.computerworld.com/action/article.do?command=viewArticleBasic&taxonomyName=Spam%2C+Malware+and+Vulnerabilities&articleId=93584&taxonomyId=85&pageNumber=1>>.
- [9] THoU, The History of Utilitarianism. 27 Mar 2009. Stanford Encyclopedia of Philosophy. 10 Apr. 2009.
< <http://plato.stanford.edu/entries/utilitarianism-history/>>
- [10] Fisher, Dennis. How long to wait?. 6 Dec 2004 eWEEK 4 May 2009
<<http://search.ebscohost.com.ezproxy.lib.calpoly.edu:2048/login.aspx?direct=true&db=afh&AN=15288134&site=ehost-live>>

- [11] Bruce Schneier, Schneier: Full Disclosure of Security Vulnerabilities a 'Damned Good Idea',
<http://www.csoonline.com/article/216205/Schneier_Full_Disclosure_of_Security_Vulnerabilities_a_Damned_Good_Idea_>
- [12] Costello Sam. Computer Security Incidents on the Rise. 14 Jan 2002 PCWorld 5 May 2009.
<http://www.pcworld.com/article/79303/computer_security_incidents_on_the_rise.html>
- [13] Witty worm proves patching 'not viable' – research. 29 Mar 2004 ZDNet 5 May 2009.
<<http://news.zdnet.co.uk/security/0,1000000189,39149962,00.htm>>
- [14] Brumley, David tracking hackers on IRC. Nov 1999 USENIX 5 May 2009.
<<https://www.usenix.org/publications/login/1999-11/features/hackers.html>>
- [15] Lemos, Robert Reverse engineering patches making disclosure a moot choice? 1 July 2005
SecurityFocus 6 May 2009. < <http://www.securityfocus.com/news/11235>>
- [16] Arbaugh, William Windows of Vulnerability: A Case Study Analysis. Dec 2000 IEEE 7 May 2009.
< http://www.cs.umd.edu/~waa/pubs/Windows_of_Vulnerability.pdf>
- [17] Larkin, Erik Protecting Against the Rampant Conficker Worm. 16 Jan 2009 PCWorld 8 May 2009.
<http://www.pcworld.com/article/157876/protecting_against_the_rampant_conficker_worm.html>
- [18] Maynor, David. The Compiler as Attack Vector. 5 January 2008 Linux Journal 5 May 2009.
<<http://www.linuxjournal.com/article/7839>>
- [19] About SecurityFocus. 5 May 2009 < <http://www.securityfocus.com/about>>
- [20] Rescorla, Eric. Security holes... Who cares? CGISecurity 8 May 2009
<<http://www.cgisecurity.com/lib/reports/slapper-report.pdf>>.
- [21] Evans David, Improving Security Using Extensible Lightweight Static Analysis. Jan 2002 IEEE 8 May
2009. < http://www.cs.virginia.edu/papers/Improving_Security.pdf>
- [22] Software Engineering Code of Ethics ACM/IEEE-CS Joint Task Force. 8 May 2009
<<http://www.acm.org/about/se-code>>.
- [23] Deontological Ethics. 21 Nov 2007 Stanford Encyclopedia of Philosophy 9 May 2009.
<<http://plato.stanford.edu/entries/ethics-deontological/>>
- [24] Zetter, Kim. Threat Level Privacy, Crime and Security Online Details of DNS Flaw Leaked Exploit

- Expected by End of Today. 22 July 2009 Wired 10 May 2009.
<<http://www.wired.com/threatlevel/2008/07/details-of-dns/>>
- [25] Kant's Moral Philosophy. 23 Feb 2004 Stanford Encyclopedia of Philosophy 10 May 2009
<<http://plato.stanford.edu/entries/kant-moral/>>
- [26] A Tend Analysis of Exploitations. 9 Nov 2000 10 May 2009
<<http://www.cs.umd.edu/~waa/pubs/CS-TR-4200.pdf>>
- [27] Arora, Ashish Timing Disclosure of Software Vulnerability for Optimal Social Welfare. Nov 2003
Carnegie Mellon University 10 May 2009.
<<http://woo.sungwhan.googlepages.com/disclosure.pdf>>
- [28] Schneier, Bruce. Security Philosophy 101: Bruce Schneier on Full Disclosure. 15 Nov 2001
Counterpane Internet Security Inc. 10 May 2009
<<http://lists.jammed.com/crime/2001/11/0117.html>>
- [29] CERT/CC Vulnerability Disclosure Policy. 10 May 2009
<http://www.cert.org/kb/vul_disclosure.html>
- [30] Geer, David Just How Secure Are Security Products? June 2004 IEEE 5 May 2009
<<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1306376&isnumber=28995>>
- [31] Weyuker, Elaine On Testing Non-testable Programs Apr 4 1982 The Computer Journal
- [32] Bugeja, Michael. "The Age of Distraction: the professor or the processor? Due to academia's
reliance on technology and the media's overemphasis on trivia, we are failing to inform future
generations about social problems that require critical thinking and interpersonal
intelligence.(FUTURE VIEW)." The Futurist 42.1 (Jan-Feb 2008): 68(2). Expanded Academic
ASAP. Gale. California Polytechnic State University. 30 May 2009
<<http://find.galegroup.com.ezproxy.lib.calpoly.edu:2048/itx/start.do?prodId=EAIM>>.
- [33] McConnell, Steve. Software Project Survival Guide. Redmond: Microsoft Press, 1997.
- [34] Federal Trade Commission Identity Theft Survey Report Sep 2003,
<<http://www.ftc.gov/os/2003/09/synovatereport.pdf>>

[34] Young, Jeffrey R. "Top 10 Threats to Computer Systems Include Professors and Students."

Education Digest 74.9 (May 2009): 24-27. Academic Search Elite. EBSCO. [Library name], [City],

[State abbreviation]. 31 May 2009

<<http://search.ebscohost.com.ezproxy.lib.calpoly.edu:2048/login.aspx?direct=true&db=afh&AN=38608383&site=ehost-live>>.

[35] SQL Injection <<http://us.php.net/security.database.sql-injection>>

[36] Siddharth, Sumit Five Common Web Application Vulnerabilities

<<http://www.securityfocus.com/infocus/1864>>

[37] Arora, Ashish Impact of Vulnerability Disclosure and Patch Availability - An Empirical Analysis

Apr 2004, Carnegie Mellon University, 31 May 2009

<<http://www.dtc.umn.edu/weis2004/telang.pdf>>

[38] Guyton L, John The Effects of Tax Software and Paid Preparers on Compliance Costs, 14 July 2005

Tax Policy Center 31 May 2009

<<http://tpcprod.urban.org/publications/urlprint.cfm?ID=1000802>>

[39] Berinato, Scott Software Vulnerability Disclosure: The Chilling Effect 31 May, 2009

<http://www.csoonline.com/article/221113/Software_Vulnerability_Disclosure_The_Chilling_Effect?page=3>

[40] Telang, Rahul and Wattal, Sunil, Impact of Software Vulnerability Announcements on the

Market Value of Software Vendors - An Empirical Investigation.

<<http://ssrn.com/abstract=677427>>