Software Vulnerabilities:

Full-, Responsible-, and Non-Disclosure

Andrew Cencini, Kevin Yu, Tony Chan

{cencini, tigeru, tonychan} @ u.washington.edu

December 7, 2005

Table of Contents

Abstract

When a software vulnerability is discovered by a third party, the complex question of who, what and when to tell about such a vulnerability arises. Information about software vulnerabilities, when released broadly, can compel software vendors into action to quickly produce a fix for such flaws; however, this same information can amplify risks to software users, and empower those with bad intentions to exploit vulnerabilities before they can be patched. This paper provides an analysis of the current state of affairs in the world of software vulnerabilities, various techniques for disclosing these vulnerabilities, and the costs, benefits and risks associated with each approach.

1. Introduction

Computer security vulnerabilities are a threat that have spawned a booming industry – between the heightened global focus on security, and the proliferation of high-profile computer viruses and worms that have had major impacts worldwide – the time is right to be in the computer security business. When one thinks about who benefits from security problems, typically the first thought would be that attackers are the primary beneficiary – breaking into vulnerable computer systems and stealing money and valuable information from victims can be an easy and profitable line of work.

However, there is another side to this burgeoning industry: the community of security professionals who build a reputation and earn a living finding and reporting security problems. While attackers stand to gain substantially from illegal activity, working as a computer security professional can be quite lucrative, with the benefit of not having to break the law or compromise one's ethics – and quite often, the technical details and challenges of this legitimate work are not much different from those when the work is done for less legitimate purposes.

This paper provides an analysis of the current state of affairs in the world of computer vulnerabilities, various techniques for disclosing these vulnerabilities, and the costs, benefits and risks associated with each approach. There are two particular bounds to be added to this discussion – the first is that this paper is scoped only to software vulnerabilities (while interesting, hardware, and physical vulnerabilities are not covered here – nor are vulnerabilities in online services, which may prove to be an interesting area of future research). The other bound placed here is that it is assumed that we are only dealing with vulnerabilities found and disclosed by 'legitimate' security researchers – that is, by those whose intent is to find and expose vulnerabilities in a lawful manner (it is, by this logic, assumed that 'illegitimate' researchers are generally unlikely to widely disclose their findings, or apply conventional ethical reasoning to such disclosures).

1.1 Structure

The first section of the paper will cover software vulnerabilities, and what are the actual and possible losses that may be incurred in the case of exploitation of such vulnerabilities. A survey of the historical record of actual attacks will be presented, as well as hypothetical examples built off of existing and possible future attack vectors. This section will provide the reader to the threat field from a cost perspective, as well as to provide actual examples to illustrate the scope of the threat.

The second section will provide an overview of the various types of vulnerability disclosure. The main classes of software vulnerability disclosure are presented, providing canonical definitions that will be used in later sections of the paper.

The third section will elaborate on the overview of disclosure types by presenting various existing and proposed practices and policies for disclosing vulnerabilities. This section brings together the first two sections by providing concrete examples of predominant disclosure practices and policies, and these

sections together, should provide enough information to introduce the fourth section which covers risks, rewards and costs of these disclosure methods.

1.2 Motivations

When discussing disclosure of software vulnerabilities, it is important to consider the motivations of those involved. The stakes are quite high in the computer security industry – being credited as the first person or company to discover a particular vulnerability is extremely important – both in finding employment and building a customer base, as it demonstrates the ability to find vulnerabilities better than others. As the ability to find vulnerabilities is a key metric that employers and customers use to measure the skill of a computer security professional or company, this situation is one of the core drivers that sets up the tricky ethical framework in the area of how one goes about disclosing vulnerabilities once they have been found.

Other motivations that security professionals and companies have, to find and disclose software vulnerabilities may be purely personal or competitive – for example, a security researcher may feel particular dislike for a software company, developer, or product, and as a result spends great time and effort searching for security flaws in that product. Researchers may also be motivated to disclose vulnerabilities because they feel that such disclosure will force vendors to be responsive in patching software and to place a greater emphasis on shipping more secure software. Finally, some researchers enjoy the intellectual challenge of finding vulnerabilities in software, and in turn, relish disclosing their findings for personal gratification or credibility from others in the field.

1.3 Terminology

Throughout this paper, several pieces of terminology are used that may have a variety of meanings – first, some definitions are provided that have been adapted from Shepherd's paper "Vulnerability Disclosure: How do we define Responsible Disclosure?"[1]

- **Product**: A software product.
- **Flaw**: A flaw in the logical operation of a product. The behavior exhibited by the flaw is such that the product is left in an undesirable state.[1] Flaws often may simply be functional in nature (for example, causing a program not to behave as specified) – but in other cases, flaws can also become security risks (see next definition).
- **Vulnerability**: A flaw becomes a vulnerability if the exhibited behavior is such that it can be exploited to allow unauthorized access, elevation of privileges or denial of service.[1] For the purposes of this paper, the terms flaw and vulnerability generally are interchangeable.
- **Exploit**: A tool or script developed for the sole purpose of exploiting a vulnerability.[1]
- **Discoverer**: The first person to reveal a flaw and determine that it is a vulnerability. Depending on how the vulnerability is discovered the discoverer may or may not be known. For example if a vulnerability is released anonymously the identity of discoverer may not be apparent.[1]
- **Originator**: The person or organization that reports the vulnerability to the vendor.[1] Note that the originator may in fact be different from the discoverer.
- **Vendor**: An entity that is responsible for developing and/or maintaining a particular piece of software. In the case of Open Source software, the "vendor" is actually a community of software developers, typically with a coordinator or sponsor that manages the development project. In the scope of this paper, the "vendor" is typically the entity (or entities) responsible for providing a fix for a software vulnerability.

- **Customer/End User:** Someone who purchases or otherwise installs and uses a piece of software. Customers are the parties that are typically the most adversely affected by exploited vulnerabilities, and are also responsible for keeping their systems patched and protected from black hat hackers.

Additionally, a few other definitions are provided for terms that are used throughout this paper:

- **Black Hat**: (or, often, "hacker") someone who finds or exploits security holes in software for malicious or illegal purposes. Rescorla[4] defines a vulnerability discovered by a black hat hacker as "discovered by someone with an interest in exploiting it."
- **White Hat**: Someone who finds or exploits security holes in software for generally legitimate and lawful purposes, often to improve the overall security of products and to protect users from black hat hackers. Alternately[4], a vulnerability discovered by a white hat hacker is described as being "discovered by a researcher with no interest in exploiting it".
- **Script Kiddie**: A non-technical "hacker" who consumes scripted exploits in order to break into other computers. Script kiddies are fairly low in the hacker food-chain; however, script kiddies can inflict real damage on real systems given the automated exploits they are provided with, which means they are more than merely an annoyance.

1.4 Timelines

There are several published timelines outlining the life of software vulnerabilities – perhaps one of the most widely accepted timelines is specified by Arbaugh, Fithen and McHugh in their paper "Windows of Vulnerability: A Case Study Analysis"[5] - which is neatly summarized by Shepherd[1] as follows:

- **Birth**: The birth stage denotes the creation of the vulnerability during the development process. If the vulnerability is created intentionally then the birth stage and the discovery stage occur simultaneously. Vulnerabilities that are detected and corrected before deployment are not considered.
- **Discovery**: The life cycle changes to the discovery stage once anyone gains knowledge of the existence of the vulnerability.
- **Disclosure**: The disclosure stage occurs once the discoverer reveals the vulnerability to someone else. This can be any disclosure, full and public via posting to Bugtraq or a secret traded among black hats.
- **Correction**: The correction stage persists while the vendor analyzes the vulnerability, develops a fix, and releases it to the public.
- **Publicity**: In the publicity stage the method of achieving publicity is not paramount but knowledge of vulnerability is spread to a much larger audience.
- **Scripting**: Once the vulnerability is scripted or a tool is created that automates the exploitation of the vulnerability, the scripting stage has been set in motion.
- **Death**: When the number of systems vulnerable to an exploit is reduced to an insignificant amount then the death stage has occurred. This can happen by patching vulnerable systems, retiring old systems, or a lack of interest in the exploit by hackers.

Rescorla[4] provides a similar summary, and notes "these events do not necessarily occur strictly in this order" – specifically, publicity and correction may occur at the same time, particularly in cases where the discoverer is the software vendor, who will also issue the patch for the vulnerability as part of the publicity. This paper largely focuses on the discovery, disclosure, correction and publicity stages.

2. Losses Due to Exploitation

Complex information and communication systems give rise to design, implementation and management errors. These errors can lead to vulnerabilities - a flaw in an information technology product that could allow exploitation.

There are several methods of classifying exploits. Exploits can be classified by the type of vulnerability they attack. For example, buffer overflow, integer overflow, memory corruption, format string attacks, race condition, cross-site scripting, cross-site request forgery and SQL injections. Today, buffer overflow related exploits remain to be the majority type.

Exploits can also be classified by how the exploit contacts the vulnerable software. A "remote exploit" works over a network and exploits the security vulnerability. A "local exploit" requires prior access to the vulnerable system and usually increases the privileges of the person running the exploit. Due to the popularity of the Internet, network-borne computer viruses and worms are the main forms of exploitations. A computer worm is a self-replicating and self-contained exploitation. It can spread with no human intervention. A computer virus requires actions on the part of users, such as opening email attachments. Viruses and worms were the most cited form of exploitation (82%). From a recent survey [14], 33% of victims recovered in one day, 30% recovered in one to seven days, and 37% took more than a week to recover or never recover.

At best, worms and viruses can be inconvenient and costly to recover from. At worst, they can be devastating. Let's look at a few recent widespread attacks [11,12,9,10] and the losses:

The Blaster, Slammer, and Code Red worms are all exploits through buffer overflow vulnerabilities. Blaster exploits Microsoft DCOM technology, Slammer exploits Microsoft SQL Server, and Code Red exploits Microsoft IIS Web Server. Figure 1 shows that, after 24 hours, Blaster had infected 336,000 computers, Code Red infected 265,000, and Slammer had infected 55,000. In both cases of Blaster and Code worms, 100,000 computers were infected in the first 3 to 5 hours. It is close to impossible for security experts to analyze the worm and warn the public. So far, damages from the Blaster worm are estimated to be at least $525 million. The cost estimates include lost productivity, wasted hours, lost sales, and extra bandwidth costs.

Exploits can also be classified by the purpose of their attack. For example, curiosity (vandal), personal fame (trespasser), personal gain (thief), and national interest (spy). With Blaster, Slammer and Code Red attacks, millions of computers were infected. However, they were probably more inconvenient and costly to recover from. Those, it turns out, may have been the good old days. Today, exploit with personal gain as the goal is the fastest growing segment.[6,7,8] These exploits can be email spam, email phishing,

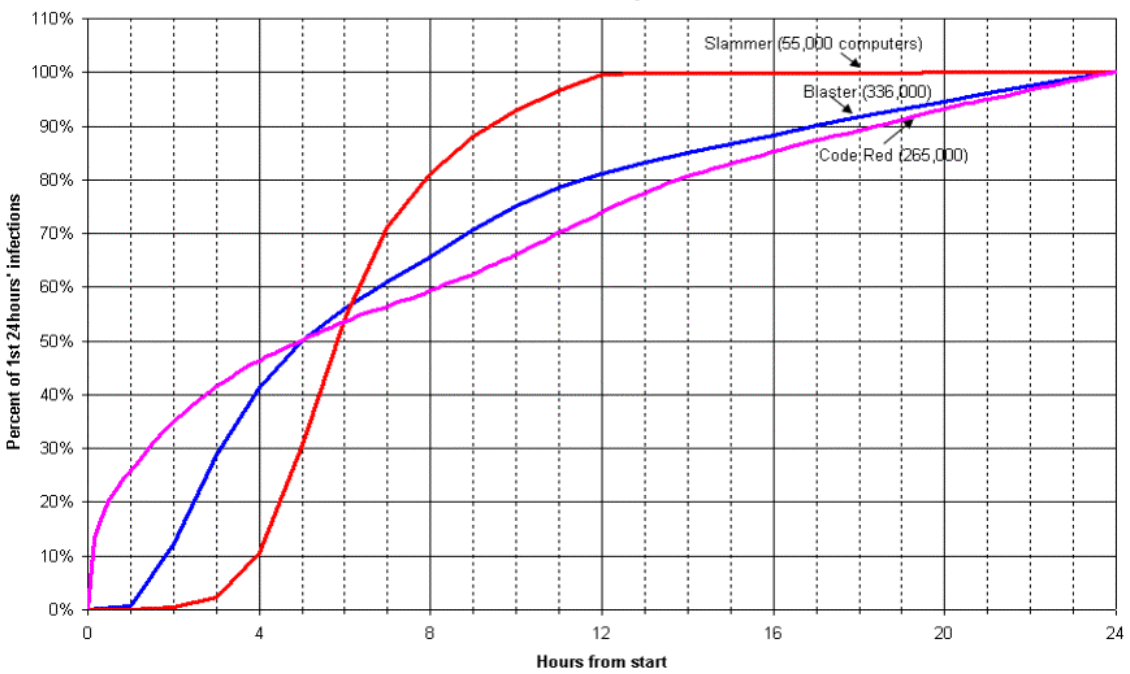

Figure 1: Blaster, Slammer, and Code Red Growth Over Day One [12]

spyware, Bots, Botnet, Keystroke loggers, identity theft, and credential theft. In these types of exploits, many people are spoofed, where over 60% visited a spoofed site, and more than 15% admitted they have provided personal data. In the U.S., 1.2 million adults have lost money due to such exploits, totaling $929 million!

3. Types of Vulnerability Disclosure

While every software vulnerability is different – from the process by which the flaw was discovered, to the way in which the vulnerability is disclosed – there are a few general categories that may be used to classify the vulnerability disclosure. There are a number of papers[1,2] in existence that define and compare various disclosure policies. The following is some background on the disclosure types being discussed throughout the paper.

3.1 Non-Disclosure

The first disclosure type is referred to as "non-disclosure." This disclosure type is probably the easiest to describe, and the hardest to quantify – in cases of non-disclosure, a security researcher discovers a vulnerability in a piece of software, and, rather than contact the software vendor or a computer security coordinating authority, the researcher instead keeps the vulnerability secret. The black hat hacker community is known for practicing a policy of non-disclosure.[1]

What makes cases of non-disclosure difficult to quantify is the paradox that there is no good way to measure how many flaws have been found, but not disclosed. There is some discussion in the work done by Havana and Röning[3] that suggests that, based on their communication models, that up to 17.3% of vulnerability findings are not disclosed; however, it remains uncertain how many vulnerabilities are discovered but remain undisclosed.

The motivations for non-disclosure can vary from malicious intent (for example, an attacker finds it to his advantage to not disclose a vulnerability so that he is able to break into numerous systems at a leisurely pace without having to worry about a patch being issued and deployed) to laziness (someone inadvertently discovers a flaw in the logic of a piece of software that lets her access supposedly protected data, but never bothers to report the vulnerability either because it is too burdensome to contact the vendor, or possibly too hard to reproduce the scenario).

There is fairly broad criticism of non-disclosure policy – major complaints take issue with the fact that systems remain unprotected while a vulnerability (and exploit) may be known, that the lack of publicity about a vulnerability may not motivate software vendors to repair the flaw in a timely manner, and that it is impossible to define a subset of "trusted" individuals who should have access to vulnerability information.[1]

Other variations on the non-disclosure method tend to have the same net end result – greater risk to users of vulnerability exploitation – for example, in some cases, a researcher may discover a flaw in a piece of software, and instead of reporting the vulnerability to a legitimate authority, the attacker will share the vulnerability (and possibly an exploit) with other hackers (essentially, "on the black market") which increases the risk to end users significantly. These types of cases, however, can tend to metamorphose into cases of full disclosure (discussed in the next section) as information spreads from the underground community into the "legitimate" world.

3.2 Full Disclosure

When a researcher discovers a vulnerability, in the full disclosure model in its purest sense (as it is defined here), the researcher informs the community at large (for example, using full disclosure methods specified by Rain Forest Puppy[17]) of the specifics of that vulnerability – how found, what software

products (and versions) are affected – and in some cases one or both of the following: how to exploit the flaw, and how to protect systems against exploitation of the flaw. There are many arguments for and against full disclosure. Advocates of full disclosure tend to argue two main points – roughly, the first point being that it is ethically correct to inform the community at large of software flaws as soon as possible (before a patch may even be issued) so that users can protect themselves by disabling the affected software (or related functionality) before an exploit is issued.[1] The second point traditionally argues in favor of full disclosure is that this tactic motivates software vendors to quickly acknowledge and patch flaws (and, presumably, for users to also quickly patch their systems) – rather than simply sit on the knowledge that a flaw exists, as can happen with other disclosure techniques (see Responsible Disclosure and its variants in the next section).[1] Intrinsically, one other benefit comes along with full disclosure, in this case, for the researcher – that is, when a vulnerability is announced immediately, the researcher gets credit immediately, without having to worry about being "beaten to the punch" by another researcher while going through a variant of the responsible disclosure process. Clearly, given the motivations and incentives of the security industry (be they professional or personal), full disclosure is a disclosure technique that can be attractive to security researchers.

On the other hand, arguments against the full disclosure method tend to parallel the arguments for full disclosure. The most salient argument made against full disclosure is that exposing a vulnerability without first consulting with a software vendor (thus allowing a patch to be developed and released) increases the risk of widespread exploitation of user computer systems – for example, many point out that within days, or even hours, following full disclosure of a vulnerability, a scripted exploit becomes available for "script kiddies" to consume.[1] This runs contrary to the argument that full disclosure protects users, because, in reality, even with heightened focus on security and automatic system updates, users are not security experts and do not follow the multitudes of security bulletins and reports that are generated on a daily basis. Additionally, while software vendors may be motivated to more quickly release software patches in cases of full disclosure, patch adoption, availability and testing may take days, weeks, or even

months in some cases (particularly in cases where a patch is being deployed across a large organization, where extensive and time-consuming testing processes are often in place) – so there exists a large window of vulnerability where a flaw is widely known amongst the security community, but the user community is unable to protect itself due to a lack of a patch (or similar issues). Finally, while full disclosure may not necessarily include exploit code for a vulnerability, this lack of code can be irrelevant as the disclosure can still make it easy for technical members of the black hat community to develop and script an exploit. [1]

There are clearly cases where it makes sense to disclose a vulnerability to the broader community for their general protection – if a software vendor is not being suitably responsive in a responsible disclosure case for example (see the next section), or in cases where a vulnerability is already fairly well known (for example, originating from the black hat community) it makes sense ethically and tactically to fully disclose the vulnerability so that users and vendors may attempt to protect themselves and fix the problem, respectively, in parallel.

3.3 Responsible Disclosure

A final type of disclosure discussed here is "responsible disclosure." Responsible disclosure falls into what many would more broadly refer to as the class of vulnerability disclosures known as partial disclosure, or limited disclosure (see Shepherd[1] for more discussion on limited disclosure, or Laakso, et al for discussion of constructive disclosure, where, amongst other things, a vulnerability disclosure is accompanied by a test suite to be used to verify future releases do not contain similar flaws).

The responsible disclosure method has many possible definitions, however, what will be used in this paper is from Stephen Shepherd's work[1] on responsible disclosure. To paraphrase, responsible disclosure is a policy in which software vulnerabilities are disclosed in a manner that puts users at the least risk

without stifling the security research community. In the simplest terms, when a vulnerability is discovered, the researcher informs the software vendor, and if the vendor is not responsive (Shepherd proposes a 30 day response deadline on an initial contact[1]), the researcher may then go to the community and proceed with, essentially, full disclosure of the vulnerability. The details of Shepherd's proposal[1] are somewhat more complicated, but the spirit of the proposal is as follows:

- Researcher discovers software flaw, and notifies the software vendor.
- If the software vendor quickly reproduces and acknowledges the flaw (and, usually, credits the person who discovered it) and develops, tests and issues a patch, the process is complete.
- If the vendor does not respond to initial contact or fails to continue communication, the originator has no option but to proceed with public disclosure without a vendor supplied patch.[1]
- In both cases, when the vulnerability is disclosed, the principles of responsible disclosure, in the Shepherd model, require that exploit code not be included with the public disclosure – while, ultimately, an exploit may be developed, there is no reason to include the exploit code in the disclosure (this also allows time for users to test and deploy the patch in their environments).

There are many partial implementations of Shepherd's responsible disclosure scheme, but there has not yet been a perfect solution developed yet that has been uniformly adopted by the computer security community. Responsible disclosure, or some variant (such as NTBugTraq[16], described later) is still an open and developing issue in the security community, but appears to be the largely preferred middle road accepted by most researchers, users, and software vendors.

4. Existing Practice, Policies and Proposals

Researchers and vendors alike share the same primary goal: reducing the risks to information systems and stopping related malicious activities. They want to inform customers and the public of vulnerabilities, but

often they have disagreements on how, when, what, and whom to disclose. The disputes are complex and there are no standards.

Organizations and individuals have proposed and conformed to various disclosure policies.[13] Six distinctive policies or proposals dated from 1999 to 2004 are chosen for examination in this report. Some of the policies support open disclosure while others support responsible disclosure. Some were designed by private sectors and some were designed by organizations funded by government. The following is a list of policies/proposals and brief introduction.

- **NTBugtraq by Cooper in July 1999.**

  The disclosure policy of NTBugtraq is one of the earliest one established in the industry and it is still in practice today. Cooper is an independent consultant specialized in Windows security.
- **Full Disclosure Policy by RFP in June 2000.**

  RFP is a security expert and its proposal advocates full disclosure policy. The policy is mainly focused on researchers.
- **Vulnerability disclosure by CERT/CC in October 2000.**

  CERT/CC is funded by government and their main goal is informing the public regarding security vulnerabilities. CERT/CC utterly opposes releasing of full vulnerability details and exploit code.
- **Responsible Vulnerability Disclosure Process by Christey and Wysopal in February 2002.**

  This proposal was an Internet-draft submitted to IETF, but it expired and was not accepted as a Request For Comments (RFC).
- **Vulnerability Disclosure Framework by NIAC in January 2004.**

  NIAC was formed by Executive Order and submitted this recommendation report to the President of the United States. The report advised building a framework in addition to defining disclosure guidelines.

- **Guidelines for Security Vulnerability Reporting and Response by OIS in September 2004.**

  OIS was formed by a group of vendors in private sectors only. It suggested responsible disclosure policy.

A summary is presented in Table 1 for comparison.

4.1 NTBugtraq by Russ Cooper

NTBugtraq was established in 1997. It is a mailing list for the discussion of security exploits and security bugs in Windows NT/2000/XP and applications running on these operation systems. There are currently more than 35,000 subscribers.[15] Russ Cooper is the founder and moderator of the NTBugtraq discussion list. [16]

In NTBugtraq, Cooper acts as a coordinator between discover and vendor. When a discoverer finds out vulnerability, he submits a report to NTBugtraq. Next, Cooper ensures the accuracy of report by reproducing the claim himself. After Cooper is satisfied with the result, he contacts the associated vendor. The vendor is then given 48 hours (excluding Saturday and Sunday) to confirm the vulnerability and a maximum of 14 calendar days to provide a fix. Once a fix is available, vulnerability information is sent to the mailing list subscribers. However, there are two clarifications in Cooper's policy. First, Cooper does not place any restrictions on the vulnerability information to be disclosed. It is because discoverers can always choose to post the information to other mailing list or newsgroups.[16] Second, throughout the entire process, discoverers can demand to disclose the vulnerability to the public at any time. Discoverers have the "ultimate call."[16]

Policy / Proposal	NTBugtraq	Full Disclosure Policy (RFPolicy) version 2	CERT/CC Vulnerability Disclosure	Responsible Vulnerability Disclosure Process (Internet-Draft)	Vulnerability Disclosure Framework	Guidelines for Security Vulnerability Reporting and Response version 2
Author	Russ Cooper	Rain Forest Puppy	CERT / Coordination Center	Christey and Wysopal	National Infrastructure Advisory Council (NIAC)	Organization for Internet Safety
Published date	Jul 1999	Jun 2000	Oct 2000	Feb 2002	Jan 2004	Sep 2004
Web Reference	http://www.ntbugtraq.com /default.aspx ?sid=1&pid=47&aid=48	http://www.wiretrip.net /rfp/policy.html	http://www.cert.org /kb/vul_disclosure.html	http://www.wiretrip.net /rfp/txt/ietf-draft.txt	http://www.dhs.gov /interweb/assetlibrary /vdwgreport.pdf	http://www.oisafety.org /guidelines/secresp.htm
Vendor should acknowledge initial vulnerability report	Within 48 hours (except Saturday and Sunday)	Within 5 working days (in respects to discoverer)	Not Applicable	Within 7 days	Within 7 business days (in respects to vendor)	Within 7 calendar days
What happens if vendor fails to acknowledge the discoverer	Vulnerability information will be disclosed immediately.	Discoverer may choose to disclose the vulnerability	Not Applicable	Discoverer should ask a coordinator to notify the vendor. If vendor is still unresponsive, coordinator should identify the best available resolution for the vulnerability	Discoverer should attempt to escalate the issue with the vendor. If it is still unsuccessful, seek for assistance from a third-party coordinator.	Discoverer sends a Request for Confirmation of Receipt to vendor. Vendor has 3 calendar days to reply. If vendor fails to reply, discoverer can get a coordinator or arbitrator for assistance.
Vendor notifies discover regarding the status updates	Not specified. Vendor should fix the vulnerability within 14 calendar days	Every 5 working days	Not Applicable	Every 7 days. Vendors may negotiate for less frequent updates	Not specified, but "vendor should keep the discoverer informed regarding progress."	Every 7 calendar days
When and what information to disclose initially	If the severity of vulnerability is low, disclosure will be released immediately. Otherwise, Cooper will wait until a fix is available from vender. However, discoverers have the "ultimate call." They may insist on releasing information immediately.	Disclose full details after a fix is ready	Disclose to the public 45 days after the initial report, regardless of the existence or availability of patches or workarounds.	Vendor should work with discoverer and coordinators to arrange a date after which the vulnerability information may be released	Discoverers should try to find a balance that will provide sufficient details without unnecessarily jeopardizing users.	Security advisories with brief information is released only after a remedy is available
When to disclose full details and exploit code	There is no limitation on the technical explanations, exploit code, or proof of concept programs	At the same time when the vulnerability is alerted to the public	Never. The number of people who can benefit from the availability of exploits is small compared to the number of people who get harmed by people who use exploits maliciously.	Vendor may ask the discoverer to allow a grace period up to 30 days, during which details that could make it easier for hackers to create exploit programs, are not released.	Never. Discoverer should withhold from any outside party any release of exploit code or detailed guide to exploiting the vulnerability when publishing advisories.	During the first 30 days of disclosure, full exploit data is shared only with people or organizations associated with defending systems against vulnerability, protecting critical infrastructures, law enforcement, etc.
Recommended communication method	E-mail with PGP, Phone	Email	E-mail through PGP or shared DES, STE/STU-III telephones, Secure FAX	E-mail	Encrypted and signed e-mail	Email

Table 1: Summary of various disclosure policies

4.2 Full Disclosure Policy (RFPolicy) version 2 by RFP

Rain Forest Puppy (RFP) is a security expert [29] and wrote the initial version of RFPolicy in June 2000. The objective of its policy was to "help establish concrete guidelines for disclosure of security problems."[17]

The RFPolicy is fairly simple. When a discoverer finds out vulnerability, he notifies the vendor. Then, the vendor should acknowledge the discoverer within 5 working days and thereafter provide status updates every 5 working days. The discoverer can disclose the vulnerability to the public if the vendor fails to respond or communicate on time. RFP favors releasing vulnerability in all aspects including exploit code. The rationale is that "other researchers are then just as likely to discover the problem and they may not bide by the guidelines set by this policy."[17]

4.3 Vulnerability Disclosure Policy by CERT/CC

CERT Coordination Center (CERT/CC) was established in November 1988. It is operated by Carnegie Mellon University and is primarily funded by the U.S. Department of Defense and the Department of Homeland Security. The objective of CERT/CC is to "analyze the state of Internet security and convey that information to the system administrators, network managers, and others in the Internet community."[19] The center offers numerous secure communication methods for the publics to send sensitive information, such as encrypted email through PGP, secure network connection through shared DES key, STE/STU-III telephones, and secure FAX.

When a vulnerability report is received, CERT/CC will forward the information to the affected vendor. Vulnerability information is disclosed to the public 45 days after receiving a report from a discoverer. The goal of CERT/CC's policy is balancing "the need of the public to be informed of security

vulnerabilities with the vendors' need for time to respond effectively."[19] Consequently, a vulnerability is published regardless of the availability of patches or workarounds after 45 days. However, the time period may be shortened if there is evidence of active exploitation or extended if major changes are required to fix the vulnerability, such as core operating system components.

CERT/CC is against of releasing exploit code. They believe that the number of people who get harmed is much larger than the number of people who can benefit from the availability of exploits.[19] However, the disclosure policy does not define any rules for discoverers and vendors.[29]

4.4 Responsible Vulnerability Disclosure Process by Christey and Wysopal

In February 2002, Steven Christey of MITRE and Chris Wysopal of @stake released an Internet-Draft to the members of Internet Engineering Task Force (IETF) and general public for commentary. The draft proposed "a formal, repeatable process for the reporting, evaluation, resolution and publication of vulnerability information."[18] Unfortunately, the proposal did not get passed and was expired after six months.

In the draft, Christey and Wysopal explained the responsibilities for all stakeholders: discovers, vendors, coordinators, and users. When a discoverer finds out vulnerability, he should notify the vendor. The Vendor should acknowledge receipt within 7 days. If the vendor is unreachable, the discoverer should get assistance from a coordinator. Upon the receipt of notification, the vendor must provide status updates to discover and coordinator every 7 days, and attempts to resolve the vulnerability within 30 days. After a fix is ready, the vendor should work with the discoverer and coordinator to arrange a disclosure date. The authors did not oppose disclosing full vulnerability details because they recognized that the security community needs the details to enhance detection tools and perform research.[18] However, to prevent

hackers from creating exploits easily, vendor can ask to postpone the release of vulnerability details up to 30 days.

4.5 Vulnerability Disclosure Framework by NIAC

The National Infrastructure Advisory Council (NIAC) was formed by Executive Order in October 2002. The council recognized that a consistent vulnerability framework could improve vulnerability management and potentially mitigate the risks to information systems.[20] Fifteen months after NIAC was established, in January 2004 NIAC finalized the report of Vulnerability Disclosure Framework. The report included specific recommendations to the President of the United States to direct the U.S. federal government as appropriate. Recommendations included aspects in various areas, such as vulnerability naming, scoring, communications, information sharing, and legal framework. Also, the report provided clear guidelines and identifies the responsibilities of stakeholders.

Naming. To reduce confusion and increase efficiency, NIAC recommended using universal naming conventions to uniquely identify vulnerabilities, similar to the project of Common Vulnerability and Exposures (CVE) by MITRE Corporation. The project assigns a common name to each vulnerability discovery so that it is easier to share data across separate databases, tools, and services among different vendors and government agencies. CVE only contains the standard name with status indicator, a brief description, and references to related vulnerability reports and advisories.[21]

Scoring. NIAC believed that a consistent threat scoring system could allow the publics to understand the severity of vulnerability.[20] The vulnerability score can assist public and private sectors to better allocate their resources and prioritize efforts to remediate those vulnerabilities with greater impacts to their systems.[20] The score can be adjusted at any time to reflect research results and active exploitation status.

Communication. NIAC recommended encrypting and signing all e-mail related to vulnerability management. Encryption can preserve the confidentiality of sensitive information, and signing can assure the original sender of message and prevent repudiation. Currently, there are products like PGP available to perform these tasks. However, acceptance is slow because many encryption products do not interoperate well. In addition, many corporations and government agencies have clear-text archive requirement and encryption is prohibited.[20] As the result, these groups of people have to send sensitive information in clear-text with the risk of being compromised or render themselves unable to contact other stakeholders who use encrypted communication.[20] NIAC recommended stakeholders to use SSL-encrypted web site for communication as one of the last resorts.[20]

Information Sharing. Information sharing is one of the key elements to protect critical infrastructures. NIAC recommended using Information Sharing and Analysis Centers Council (ISACs) as the channel for sharing information on vulnerabilities and their solutions.[20] Besides sharing, NIAC emphasized that the accuracy of information is very important. Inaccurate vulnerability reports can distract vendors and service providers from their primary operations and may unfairly damage their reputation. However, vendors should try their best effort and not to deny a report until they are positive that the report is inaccurate. Vendors should not threaten discoverer who reports vulnerabilities with legal actions since this will undermine the economy, businesses, and citizens.[20]

Legal Framework. Today, some stakeholders hesitate to disclose vulnerability because they are fearful of potentially violating laws and incurring financial liabilities or reputation injury. For example, in July 2005, Michael Lynn planned to disclose the details of a vulnerability in Cisco's Internetwork Operating System (IOS) at a Black Hat conference in Las Vegas. Cisco Systems attempted to stop his presentation by filing a retraining order by U.S. District Court.[22] For another example, in 1999, under the law of the Digital Millennium Copyright Act (DMCA), DVD Copy Control Association (CCA) and the Motion Picture Association of America (MPAA) filed a lawsuit to prevent Jon Johansen from publishing codes

that can circumvent the Content Scrambling System (CSS).[23] Again in August 2002, Hewlett Packard attempted to use DMCA and computer crime laws to prosecute Secure Network Operations (SnoSoft) who disclosed vulnerabilities in HP TruUnix Operating System.[24,25] Although the United States has the Freedom of Information Act (FOIA) for disclosing information, vulnerability disclosure is exempted from the law.[20] NIAC suggested the U.S. government to review and reform the legal framework and public policy so that stakeholders can share information without the fear of financial or other liability.

Guidelines. Under NIAC's recommended framework, discoverers should "find a balance that will provide sufficient details without unnecessarily jeopardizing users" when publishing advisories.[20] NIAC opposed releasing exploit code and complete guides to exploiting the vulnerability. Vendors should acknowledge the discoverer within seven business days after receiving a report. A third-party coordinator should be used for assistance if a discoverer cannot contact the vendor. End users and organizations should employ deployment and mitigation plans after they have received notification of the vulnerability and corresponding fix.

4.6 Guidelines for Security Vulnerability Reporting and Response ver. 2 by OIS

The Organization for Internet Safety (OIS) was formed by a group of vendors including @stake, ISS, Microsoft, Oracle, SGI, and Symantec in September 2002.[27] Surprisingly, researchers and end users are not invited to join.[28] OIS believed that "the industry should self-regulate" and does not support a federal law codifying the disclosure process or any kinds of mandates.[27] The document of Guidelines for Security Vulnerability Reporting and Response provided step-by-step instructions and flow charts illustrating what stakeholders should perform during each phrase of the vulnerability window.

In OIS' guideline, when a discoverer finds a vulnerability, he should send a Vulnerability Summary Report (VSR) to the vendor. The vendor is then required to acknowledge receipt of the report within

seven calendar days and thereafter provide updated status every seven calendar-days. Although the vendor is given a conventional 30 calendar days to fix vulnerability, OIS noted that the thoroughness of investigation and high quality of fix are also important in addition to speed.[27] Vendors and discoverers should release vulnerability information to the public only after a remedy is available. To protect critical infrastructures, exploit code and detailed data are shared only with people or organizations associated with defending systems (i.e. intrusion detection and anti-virus vendors) during the first 30 days. The 30 days grace period may be shortened if the vulnerability becomes actively exploited.

During the entire process, if a discoverer does not receive a response, they can remind the vendor by sending a Request for Confirmation of Receipt (RFCR). The vendor must respond to a RFCR within three calendar days. If the RFCR fails, the discoverer can get assistance from a third-party coordinator to facilitate the communication between discoverer and vendor. An arbitrator can also be employed if a disagreement occurs. However, there are several preconditions in OIS' guideline. The coordinator and arbitrator can be used only if both the vendor and discoverer mutually agree. In addition, both the vendor and discoverer must have a mutual consent on the scope of authority and duty for the coordinator and arbitrator. This mutual agreement could be challenging to accomplish in real practice.

5. Risks, Rewards and Costs

5.1 Costs and Risks

In some cases, disclosure of software vulnerabilities can help improve the overall security of computer systems, while in others, such disclosures can lead to costly widespread exploitation of security problems by black hat hackers. In any case, discovery of a software vulnerability is not free: costs are incurred by software vendors to process, verify, fix, and distribute patches for a vulnerability; researchers spend

considerable time and effort in locating software flaws; and users and administrators spend large amounts of time finding, testing and deploying patches to their systems.

Rescorla[4] proposes a cost model for software vulnerabilities that is general enough to span the spectrum of actual possible losses that may be incurred as part of the software vulnerability cycle. In his model, the costs are largely user-costs – centered around what it costs to apply fixes for a given vulnerability, as well as the costs incurred in case of an exploitation (not included are costs to vendors in actually fixing flaws, as well as the costs to a vendor's reputation in high-profile exploitation cases). The model states that, for the task of applying patches, there is a cost incurred that is roughly linear with the number of machines that are patched; risks (of intrusion) are incurred in the period when a vulnerability is known about privately but not publicly; and there are actual costs and risks to un-patched systems when a disclosure has been made (in any case of intrusion of a single system, the cost scales with the value of the system – but for wide-scale intrusions, like worms, the costs may become significantly higher due to other collateral damage).

5.2 Cost-Benefit Analysis

An excellent cost-benefit analysis of disclosure is proposed[1] where the question of whether or not to disclose a vulnerability is posed, based on the short and long term implications of the disclosure. Since actual costs of exploitation (large- and small-scale) can vary so much and can be so hard to measure (see discussion earlier in this paper on actual incidents), an abstract model is used to form the decision matrix of whether or not to disclose. For a discovered vulnerability, the choice to disclose only reduces the expected cost of intrusions based on the formula[4]:

$$P_r (C_{priv} + C_{pub}) > C_{pub}$$

where:
1. P_r is the probability that a vulnerability will be rediscovered.
2. C_{priv} is the cost of private exploitation (where a vulnerability is exploited by black hat hackers without being known about publicly)
3. C_{pub} is the cost of public exploitation (where a vulnerability is exploited, but has also been disclosed)

In other words, if a vulnerability is to be disclosed, the additional cost of private exploitations by black hat hackers must be greater than the cost of exploitations incurred by disclosing the vulnerability in the first place.[4] So, by this logic, if a researcher discovers a flaw in a product, the only reason she should make use of the full disclosure policy would be in cases that not disclosing the vulnerability immediately puts users at greater risk in the short term than would be incurred as part of disclosure performed in the longer term (presumably, after a vendor can generate a patch).

Relative to the probability of vulnerability rediscovery, Ozment[30] also provides some background, looking at what value is provided by the vulnerability discovery process in general.

5.3 Non-Disclosure

When discussing the risks and rewards of a non-disclosure policy, it is clear (as discussed earlier) that the benefits of non-disclosure tend to be limited to black-hat hackers who discover and exploit vulnerabilities, but do not disclose them – and benefits can extend as well to software companies whose reputations are not tarnished by the disclosure of security flaws. Vidstrom[2] provides examples of "fake" arguments that are commonly made in favor non-disclosure policies – arguments such as:

- **Money.** The vendors simply think they will make more money from keeping the vulnerabilities secret. The web site of Ntsecurity.nu performed a poll on question: "Do you think that software vendors deliberately neglect security to increase short-term profit?" The result was 87% Yes, 7%

No, and 6% Not Sure. If we don't trust the vendors, we need some kind of balancing force - for example full disclosure. [2]

- **Control.** "If I keep the information secret I will be in control, me and my elite security expert friends will not allow anybody else to enter our closed elite group."[2]

The main losers in the case of non-disclosure of vulnerabilities are users – in all cases, as described previously, there is a real risk to all users when a security flaw is known but not disclosed to software vendors – as such, the costs and risks of non-disclosure far outweigh the perceived benefits of non-disclosure.

5.4 Full Disclosure

In the cost-benefit analysis model from Rescorla[1], above, it is asserted that the policy of full disclosure is only less costly than that of partial or responsible disclosure when the immediate short-term risks of private exploitation are greater than the longer-term risks of public exploitation.

Some of the purported benefits of full disclosure for a vulnerability are that the vendor is motivated to provide a patch or workaround in a timely manner, that an administrator might make use of exploit code to test for the existence of vulnerable systems, or to test the integrity of a patch that has been distributed to correct a vulnerability. [1] However, the main benefits in many full disclosure cases are realized primarily by the security researchers who found the vulnerabilities – on this point, Vidstrom[2] contends, again, that the personal fame associated with such a disclosure is a non-argument because of the potential risks being incurred on the general public as a result of that disclosure.

There are many risks that offset the benefits of full disclosure – for example, a policy of full-disclosure arms hackers with pre-made exploits for attacking systems – with arguments being made that hackers are less likely to spend time discovering new vulnerabilities. Additionally, as has been stated several times previously, full disclosure of a vulnerability without giving a vendor time to release a fix puts all users at risk (see the cost benefit model above). On the whole, the risks and costs incurred as part of full disclosure (in its purest sense) outweigh the benefits.

5.5 Responsible Disclosure

The benefits of responsible disclosure (and many other "partial disclosure" policies) are that the risks to end users (and thus, the costs incurred by end users) tend to be the smallest. While difficult to define a trusted set of individuals in the most perfect sense, responsible disclosure keeps information about a vulnerability within that trusted set of individuals until a patch is released, and when disclosure does take place, the full technical details of the vulnerability are only provided once a patch has been provided by the vendor.[1] Essentially, the path of responsible disclosure tends to be the lowest-cost, highest-benefit path that vulnerability discoverers can take in many cases using Rescorla's cost-benefit model. Aside from the limitations factored into that model, other risks of responsible disclosure can be that vendors may not be motivated to repair flaws in a timely manner (thus expanding the risk window of private exploitation by black hats), potential liability issues for security researchers releasing information to software vendors, and the general re-discoverability problem (see Ozment[30]) – on the whole, however, forms of responsible disclosure tend to be one of the most widely agreed upon best practices, with the fewest standard implementations[1] (NTBugTraq[16] being one of the more widely accepted forms).

6. Conclusion

The Internet will continue to grow and change the role that software plays in our lives. As our lives depend more and more on the Internet and software, security becomes essential. When software

vulnerabilities are discovered, it is in the public interest that existing systems with vulnerabilities are being fixed in a timely fashion. The question is that when vulnerabilities are discovered, how discoverers should disclose them. If discoverers disclose vulnerabilities publicly with exploitation details, script kiddies or black hats can use the same information to launch attacks. If discoverers do not disclose vulnerabilities publicly, vendors have less motivation to fix their software and provide patches. Among debates of disclosure policies, responsible disclosure policy tends to be one of the most widely agreed upon best practices. However, the biggest challenge facing any new vulnerability disclosure policy is universal adoption.

At the same time, while it is necessary to set policies for responsible disclosure, it could make all other forms irresponsible. When a disclosure policy becomes adopted, it could be a small next step to pass legislation criminalizing all other "irresponsible" disclosure. It is also a valid concern that vendors could use legal actions to prevent disclosure of vulnerabilities, such as the cases of Cisco and Michael Lynn, and HP and SnoSoft (as pointed out in section 4.5).

The Internet has brought the software industry to a global level. Vulnerability disclosure policy is not one that a single nation can govern. For example, if the U.S. passes legislation for software vulnerability disclosure, it won't necessarily apply to Russia. A global approach towards adoption of the new policy is the best strategy. Instead of relying on laws, we should apply economic principals when thinking of the disclosure policy, which should motivate both the discoverers and the vendors.

Ultimately, we need to have guidelines for a reasonable course of action for disclosing software vulnerabilities. However, it is too early for the security community to understand the problem enough yet to set a single enforceable vulnerability disclosure policy. The debate should continue. And let it continue!

Reference

1. S. Shepherd, "Vulnerability Disclosure: How do we define Responsible Disclosure?" SANS Institute, http://www.giac.org/practical/GSEC/Stephen_Shepherd_GSEC.pdf, Feb 2003.

2. A. Vidstrom, "Full Disclosure of Vulnerabilities – Pro/Cons and Fake Arguments," Net Security, http://www.net-security.org/article.php?id=86, Apr 2002.

3. T. Havana and J. Röning, "Communication in the Software Vulnerability Reporting Process", MA thesis, University of Jyvaskyla, http://www.ee.oulu.fi/research/ouspg/protos/sota/FIRST2003-communication/paper.pdf, June 2003.

4. E. Rescorla, "Is Finding Security Holes a Good Idea?" RTFM Inc, http://www.dtc.umn.edu/weis2004/rescorla.pdf, Jul 2004.

5. W. Arbaugh, W. Fithen, and J. McHugh, "Windows of Vulnerability: A Case Study Analysis," IEEE Computer Society Press, http://www.cs.umd.edu/~waa/pubs/Windows_of_Vulnerability.pdf, Dec 2000.

6. S. Wildstrom, "Probing Your PC's Weak Spots," BusinessWeek Online, http://www.businessweek.com/technology/content/may2005/tc2005052_2731_tc024.htm, May 2005.

7. S. Wildstrom, "Clicks that Make PCs Sick," BusinessWeek Online, http://www.businessweek.com/technology/content/may2005/tc20050510_5936_tc205.htm, May 2005.

8. S. Wildstrom, "Viruses Get Smarter and Greedy," BusinessWeek Online, http://www.businessweek.com/print/technology/content/nov2005/tc20051122_735580.htm, Nov 2005.

9. Wikipedia, "Timeline of Notable Computer Viruses and Worms," http://en.wikipedia.org/wiki/Timeline_of_notable_computer_viruses_and_worms, Nov 2005.

10. SANS Institute, "The Twenty Most Critical Internet Security Vulnerabilities Version 6.01," SANS web site, http://www.sans.org/top20, Nov 2005

11. D. Litchfield, "Unauthenticated Remote Compromise in MS SQL Server 2000," NGSSoftware Insight Security Research Advisory, http://www.nextgenss.com/advisories/mssql-udp.txt, Jul 2002

12. R. Pethia, “Computer Viruses: The Disease, the Detection, and the Prescription for Protection,” CERT/CC, http://www.iwar.org.uk/comsec/resources/virus-nov-06-03/Pethia1784.htm, Nov 2003.

13. University of Oulu Secure Programming Group (OUSPG), “Vulnerability Disclosure Publications and Discussion Tracking,” http://www.ee.oulu.fi/research/ouspg/sage/disclosure-tracking, May 2005.

14. R. Richardson, “2003 CSI/FBI Computer Crime and Security Survey,” Computer Security Institute, http://www.security.fsu.edu/docs/FBI2003.pdf,

15. TruSecure Corporation, “TruSecure’s Research” http://www.trusecure.com/knowledge/research, Dec 2005

16. R. Cooper, “NTBugtraq Disclosure Policy,” NTBugtraq, http://www.ntbugtraq.com/default.aspx?sid=1&pid=47&aid=48, Jul 1999

17. Rain Forest Puppy, “Full Disclosure Policy (RFPolicy) v2.0,” http://www.wiretrip.net/rfp/policy.html, Jun 2000.

18. S. Christey and C. Wysopal, “Responsible Vulnerability Disclosure Process,” IETF draft, http://www.wiretrip.net/rfp/txt/ietf-draft.txt, Feb 2002.

19. CERT Coordination Center, “CERT/CC Vulnerability Disclosure Policy,” http://www.cert.org/kb/vul_disclosure.html, Oct 2000.

20. J. Chambers and J. Thompson, “Vulnerability Disclosure Framework,” National Infrastructure Advisory Council, http://www.dhs.gov/interweb/assetlibrary/vdwgreport.pdf, Jan 2004.

21. MITRE Corporation, “CVE Frequently Asked Questions,” http://www.cve.mitre.org/about/faq.html, Oct 2005.

22. J. McCormick, “Researcher Heats up Black Hat Conference with Controversial Cisco Presentation,” TechRepublic, http://techrepublic.com.com/5100-1009_11-5810871.html, Aug 2005.

23. S. Pruitt, “Norway Indicts Teen Creator of DeCSS,” IDG News Service, http://www.pcworld.com/news/article/0,aid,79285,00.asp, Jan 2002.

24. J. Schiller, “Responsible Vulnerability Handling: ‘A Hard Problem’,” Secure Business Quarterly, Vol 2, Issue 3, http://www.sbq.com/sbq/vuln_disclosure/sbq_disclosure_hard_problem.pdf, Sep 2002.

25. K. Zetter, "HP, Bug-Hunters Declare Truce," PCWorld, http://www.pcworld.com/news/article/0,aid,103853,00.asp, Aug 2002.

26. Organization for Internet Safety, "Guidelines for Security Vulnerability Reporting and Response," http://www.oisafety.org/guidelines/secresp.html, Sep 2004.

27. Organization for Internet Safety, "About Organization for Internet Safety," http://www.oisafety.org/about.html , Dec 2005.

28. D. Verton and J. Vijayan, "When Is It Safe to Disclose Security Flaws? Industry Group Sets New Guidelines for Reporting Software Holes," ComputerWorld, http://www.pcworld.com/news/article/0,aid,111879,00.asp, Aug 2003.

29. Y. Tian, "Regulation for Reporting Security Flaws," Helsinki University of Technology, http://users.tkk.fi/~tianyuan/slides/template, Nov 2002.

30. A. Ozment, "The Likelihood of Vulnerability Rediscovery and the Social Utility of Vulnerability Hunting," University of Cambridge, http://infosecon.net/workshop/pdf/10.pdf, Jun 2005.