Chapter 6
Responsibility for the Harm and Risk of Software Security Flaws

Cassio Goldschmidt
Symantec, Corp., USA
Melissa J. Dark
Purdue University, USA
Hina Chaudhry
Purdue University, USA

ABSTRACT
Software vulnerabilities are a vexing problem for the state of information assurance and security. Who is responsible for the risk and harm of software security is controversial. Deliberation of the responsibility for harm and risk due to software security flaws requires considering how incentives (and disincentives) and network effects shape the practices of vendors and adopters, and the consequent effects on the state of software security. This chapter looks at these factors in more detail in the context of private markets and public welfare.

KEYWORDS: Software security, common good, incentives, externalities, secure coding, patching, risk, vulnerability disclosure.

INTRODUCTION
This chapter describes the current landscape of the responsibility for the harm and risk of software security flaws. We focus on software vulnerabilities for several reasons. Software assurance is critically important to information assurance and security and we believe it will be important for some time to come. While improvements in software security will be made, these will be incremental at best. Getting software right is still an art. No practical, formal methods exist to prove application security nor does a definitive authority exist to assert the absence of vulnerabilities. Small coding errors can lead to fatal flaws due to interactions among different components of complex software. The first portion of this chapter outlines who vendors are, their current practices to securing software, and overviews the forces that impinge on vendors’ software security practices.

While software is developed by vendors, it is deployed, operated, and sometimes adapted, by a myriad of adopters. Numerous decisions that adopters make have implications for the state of software security, for example, installation with default settings or patching practices. Given the interdependent nature of information systems, the role of adopters, their practices, and the forces that constrain their software security practices are also discussed. Special attention is given to how current practices in patch availability and deployment affect software security.
Despite best effort to build, deploy, and govern secure software, some portion of software vulnerability is inevitable, which brings us to the role of vulnerability disclosure. Vulnerability disclosure is about the sharing of vulnerability information: relevant issues include how, when, with whom, and how often vulnerability information is shared. This chapter discusses the role of vulnerability disclosure on the responsibility for harm and risk of software insecurity with a focus on how disclosure enables and constrains software security practices.

Producing robust software that is able to withstand attacks, work around hardware limitations, and even inform users about the potential security risks related to their choices is no longer seen by society as nice to have, it has become a requirement. Enacting this mandate, however, is far from clear. Improving software security is as much about economics, public policy, and social welfare as it is about abuse cases, error conditions, and testing methodologies. Who should be responsible for the harm and risk caused by security flaws?

BACKGROUND

One of the challenges in understanding who should ultimately be responsible for the harm and risk caused by security flaws is our lack of a full understanding of the nature of information technology risk. “As systems become more complex and interconnected, emergent behavior (i.e., unanticipated, complex behavior caused by unpredictable interactions between systems) of global systems exposes emergent vulnerabilities” (Computing Research Association, 2003, pg. 21). This complexity and emergence make risk assessment hard. Our existing mathematical/statistical risk models are based on independent failures, where “a component failure in one part of the system does not affect the failure of another similar component in another part of the system. This leads to especially beautiful and useful models of system failure that are effectively applied thousands of times a day by working engineers” (Computing Research Association, 2003, pg. 21). Unfortunately, these models are not transferrable to networked systems where failures are interdependent, not independent.

We need models that can account for dependencies between system components in a manner that sheds light on how the behaviors of system components interact to lead to system failure. Progress in interdependent risk measurement will enhance the effective management of investment. “Without an effective model, decision-makers will either over-invest in security measures that do not pay off or will under-invest and risk devastating consequences” (Computing Research Association, 2003, pg. 21). Interdependencies also pose considerable challenges when it comes to assigning liability, and formulating reasonable policy and associated compliance.

Despite our lack of understanding of the nature of interdependent risk, it is widely acknowledged that we are interlinked and the risk interdependent. Interdependent risk necessitates interdependent responsibility. In the words of Jane Addams (1910), “the good we secure for ourselves is precarious and uncertain, is floating in mid-air, until it is secured for all of us and incorporated into our common life” (pg. 116). Addams was awarded the Nobel Peace Prize in 1931 for her unwavering commitment to social improvement through cooperative efforts.

The relevance of her words today is in conceiving solutions for how self-interested individuals and groups collectively and cooperatively work toward improved software security. Rational choice suggests that humans, individually, and in groups, act in their own best interest. The logic of collective action states that large groups with a common interest will not act collectively in the absence of individual incentive or compulsion (Olson, 1971). When the good to be provided is a public (shared) good, by definition, the good is of benefit to a group larger than those who are providers of the good. Any individual or group will only get a portion of the benefit of any expenditure made to obtain the good. When circumstances are
such that the gain to an individual (or small group) could be disproportionate to the immediate losses or costs, there will be a failure to take collective action as each individual perceives the possibility of losing more than individually gained (as well as perceiving that someone else gained at lesser cost). The result will be a failure to provide the public good. Given the premise of self-interest, the individual or group will seek to discontinue the expenditure before the optimal good for the group has been obtained.

This suggests that the question of the responsibility for harm and risk from software security flaws requires looking at the groups involved and their respective interests. It also requires that we consider the dual private good/public good nature of software. While software has been privatized, software vulnerabilities have largely been treated as externalities; which means that the costs of software security are borne by those who are not involved in the transaction. Given these challenges, who should be responsible for the provision of public goods, to what degree, and how becomes, in part, the purview of government. This chapter considers the responsibility for harm and risk of software security flaws in the context of collective action, competing private interests, and public welfare.

The Common Good

While an exhaustive definition is beyond the scope of this chapter, a working definition of philosophical notions of the common good is in order, as well as discussion of the relationship of common good, public policy, and economics. This foundation should serve as a framework so that when we discuss current issue in software assurance and software vulnerabilities, we can go beyond recounting current debates to consider theoretical underpinnings providing an analytical basis from which we can discuss the consequent implications.

In the abstract, a good is a common good when the causes are also the ends. That is, the internal order is a good in and of itself, and is also an instrumental good that serves as a means to other ends. The “common” is comprised of particulars, where particulars are entities or functions that correlate to the whole. For example, an individual is part of a family, a family is part of a community, a community part of a city, a city part of a state, and a state part of a nation. The whole to which they correlate though is not merely the sum of these parts and their utilities. Common goods are more than a collection of particular goods; the whole of the common good is more than the sum of its parts. The goodness of the whole exists independent of the goodness of each part, yet the goodness of the whole also exists to enable the goodness of constituent parts so that the parts can in turn constitute the whole.

The ‘common good’ can be traced to the early foundations of political theory. In Aristotle’s Politics, Aristotle considers what form of community is best for all to realize the ideal of their life. According to Aristotle (Aristotle, 350 BC), a common good state concerns itself with the relative equality of outcomes where all citizens can flourish. Aristotle’s proposed that the polis (city in Greek; the modern day equivalent would be the nation state) should be a context in which citizens deliberate about the common good in a manner that it, in and of itself, is a common good. The ideal is that the polity should be a common good itself as well as exist to ensure that other goods are realized by the communis (Latin for public) to include both shared and individual resources for collectives and individuals. The polity includes the actions, practices, and institutions that regulate social and political life such that these actions, practices and institutions serve the good of all its citizens and not only the good of some. The polity includes social systems, institutions, and environments on which we all depend and work in a manner that benefits all people. For example, the systems might include the public health care system, the public safety and security system, the legal and political system, and the economic system.

While the notion - the ideal - of the common good is transcendent, how the material and manifest work of the common good are worked out is a matter of collective action, including the action of particular groups and the collective action of the totality of the groups. It is through this lens that we will consider
responsibility for harm and risk of software security flaws. As we explore the contours of software assurance, we consider three constituent parts, vendors, adopters, and vulnerability disclosure, as well as the sum of those parts.

Welfare Economics

Welfare economics is a branch of economics that looks at economic welfare theory with regard to public policy questions, which are by definition questions of the common good. The objective of welfare economics is to help society make better choices with regard to the use of resources. Better choices alludes to the criteria for how goods and services produced in an economy should be distributed among individuals and considers issues of liberty, justice, and equity from political, moral and economic perspectives. Welfare economics focuses on using the scarce resources optimally such that the maximum well-being of individuals in society is realized.

Welfare economics seeks to define social welfare (the total well-being of the entire society) in economic terms that describe the activities of individuals and collectives. However, because the welfare of individuals and collectives cannot be directly observed, an abstract measure of welfare, called a utility function, is used. The utility function in welfare economics can be understood by contrasting it to the notion of profit maximization in microeconomics. Whereas microeconomics is concerned with how businesses provide goods and services to maximize profit, utility maximization is concerned with what resources should be allocated to maximize “satisfaction” or “happiness” within a society and how. The idea is that utility within a society can be increased and decreased; and, those shifts can be represented in economic terms as social welfare gains or losses.

Utility within a society includes goods and services that can be purchased in the marketplace as well as goods and services that private markets will typically not provide, which are referred to as common goods (also known as public goods and collective goods in economics) and club goods. In this chapter, we call them public goods so as to not confuse them with “the common good”. A public “good” refers to a thing, a commodity, a state, or a service. Examples of public goods are roads, defense, law enforcement, air, and information goods, such as software. By calling something a public good, we are not implying that it is “worthy”, “desirable”, or “satisfactory”. Public goods can actually be “bad”. For example, pollutants are public goods. However, the reciprocal – pollution abatement – can be called the public “good” in both senses, the thing and the value.

A good is considered a public good when the cost of providing the good to another user is effectively zero, that is, there are no marginal costs. Public goods are also defined as goods that are non-rivaled and non-excludable. A non-rivaled good means that a person can benefit from the good without denying another person the opportunity to benefit from the same good. So, in the case of roads, my getting to drive on a well-kept road is a benefit. It does not cost any more to provide this good to you and me than it does to provide the good to only me (marginal costs are zero). And my opportunity to benefit from this good does not deny you of the same benefit (the good is non-rivalrous). Non-excludable means that no one can be excluded from using the good or service regardless of whether or not they pay for the service. For example, a street light is non-excludable. If I invest money to install a street light outside my house for safety purposes, I cannot prevent my neighbor from the safety benefit of the light. The cost of my neighbor having light is no more than the cost of me having light. This is where the case of software becomes interesting. The marginal costs in producing software are effectively zero. It is possible for a software vendor to provide me with a piece of software and provide you the same piece of software at no additional costs. However, digital rights protection is often applied to make software rivalrous. Just as the process of converting source code into binary code makes it excludable.
Public goods are sometimes thought of as market failures. In a market system, goods, services, and information are exchanged forming part of the economy. This exchange of goods and services for money is called a transaction. Transactions are sources of information; they tell us what consumers pay for products at given quantities, and the price for which producers sell given quantities of products. Markets enable tradable items to be evaluated and priced. The two fundamental forces of a market are supply and demand. A market that is in equilibrium produces a good/service at a price where the quantity demanded by consumers equals the quantity supplied by producers. Such a market is considered to be in equilibrium and efficient.

A market failure refers to circumstances when market-like behavior fails to produce efficient results; such instances are commonly referred to as externalities. Externalities arise when the well-being of a consumer or the production possibilities of a producer are affected by the action of another decision maker in the economy and this interaction is not attributable to changes in price. Public goods, then, can be thought of as externalities – public goods (or “bads”) arise in situations where a competitive market fails to allocate resources efficiently. The primary reason we see market failures of public goods stems from their non-excludable nature. When it is not possible or overly costly to make a good excludable, people will free ride. Free riders are individuals who have access to consume the good, but will not pay for it. Because free riders decrease private firm profits, private markets fail to produce public goods at socially optimal levels. When a market fails to produce a public good at the level desired, other mechanisms are used to reach socially optimal production levels. These include civic responsibility, volunteerism, private donation, and public provision (policy). Throughout this chapter we discuss the ways that market failures affect the responsibility for harm and risk of software security flaws.

The traditional means of addressing externalities through public provision are mechanisms such as taxation and quotas. Positive taxes impose a cost on the producer for producing the externality or a consumption tax on the consumer, both of which aim to decrease the externality. Negative taxes, also called subsidies, seek to subsidize the provision of a public good in the private market by incentivizing the producer to produce less or the consumer to consume more of the externality. Later in this chapter we talk about policies for providing incentives for software patching, two of which are patching rebates (a subsidy) and usage tax.

One of the most widely used theoretical concepts (though not undisputed) in welfare economics is the “Pareto Criterion”, proposed by Vilfredo Pareto in 1896 (Just, Hueth, and Schmitz, 2005). The idea of the Pareto Criterion is that a situation is socially desirable when it is possible to make a change to make one (or some) person(s) better off while no one is made worse off. A policy change that makes at least one person better off by moving them from state A to state B while also ensuring that no one is made worse off would be a pareto improvement. This does not mean that no one loses consumption or access to goods and services. Some portion of the population could experience loss, but through compensatory mechanisms their loss could be offset so that, in effect, overall welfare remains unchanged. If two alternatives exist to the status quo, the alternative that produces the most social gain is considered pareto superior. While a thorough discussion of the idea of pareto optimality and criticisms thereof are beyond the scope of this chapter (interested readers should have no problem finding additional resources), as we shall see throughout this chapter, questions of pareto improvements and pareto superiority, whether explicitly or only implicitly, are at the core of the debate regarding responsibility for risk and harm of software security flaws.

We turn now to discussion of the software assurance milieu. We look at the players; their roles, and their practices in the context of public good; incentives and disincentives; and externalities in an effort to shed light on the complexity of responsibility for harm and risk of software security flaws. Given that there are no easy answers and that scrutiny frequently produces more questions before it produces any answers, we conclude with a list of questions that we think will be important in the future.
VENDORS

Generally speaking, software vendors are companies that specialize in making software. Most of us are familiar with large software companies, such as Google and Microsoft. However, there are countless companies that produce software for a myriad of diverse purposes; administering medication, controlling inventory levels, routing packages, and scheduling and coordinating air transportation are just a few. All of these entities are software vendors.

Vendor Practices Regarding Software Security

As the creators of software applications, vendors generally bear accountability for the safety and functionality of their products. A generally accepted practice is for software vendors to stay abreast of common attacks against software applications and actively design products that reasonably withstand and mitigate the impact of such attacks. However, the difficulty - and considerable uncertainty - comes from the fact that it could be the poor operation of software that causes harm; in which case the vendor should not be responsible for the misuse of an artifact they developed. In some ways, that would be like holding automobile manufacturers responsible for people’s driving habits. A software-related computer failure has several parties who may be partially responsible: the software vendor, the computer vendor, the network vendor, the user, possibly another hacker, and so on (Schneier, 2008). To date, models of partial responsibility in this area have been elusive.

Despite uncertainty about what it means for a vendor to “bear accountability” for a software application, there are best practices that vendors can, and do, use to improve software security. A systematic engineering methodology can improve the overall quality of the product and reduce security risk. Best practices prescribe that security should commence when product ideas are conceived and be an intrinsic part of the software development process. Threat analysis should be performed early in the development lifecycle to highlight problems that are more architectural in nature. During threat analysis, seasoned engineers step into the shoes of attackers to construct threat models. The software architecture is then analyzed and scrutinized to search for possible abuse cases. The data flow diagrams created in this phase of the development serve as the basis for security test cases later performed during the development lifecycle.

Because poor coding practices can result in software that is vulnerable to cyber attacks, another best practice of vendors is the utilization of well-defined policies for their coding efforts. Programming library functions that have been rendered obsolete due to safety, but are still supported for compatibility reasons, should not be utilized when crafting new code. Whenever possible, these functions, also known as deprecated functions, should also be replaced with the new and safer counterparts. Developers are also encouraged to avoid “reinventing the wheel” and instead make use of security patterns, which are well-understood solutions to a recurring information security problem. Vendors should utilize new programming languages and compilers that significantly simplify the developer’s task of writing secure code by providing sophisticated memory management capabilities and built-in mitigations against common attacks. Vendors should avail themselves of the current state of the art in static and dynamic source code analysis tools to scan source code for security flaws and partially automate vulnerability testing. Static source code analysis can be performed without the need to execute programs. In most cases the analysis is performed on some version of the source code and in the other cases it is performed on some form of the object code. In contrast, dynamic source code analysis identifies vulnerabilities in a runtime environment.

However, because neither static nor dynamic analysis can recognize sophisticated attack patterns or business logic flaws, another best practice among security conscious vendors is to perform manual code reviews. Many vendors go beyond the use of automated tools and engage security experts to perform a
special type of security assessment called penetration testing. In these assessments, security experts simulate hacker attacks against the target system to find security flaws. In some cases, vendors hire third parties to conduct the penetration testing in order to obtain unbiased assessment of their products.

Software is developed by people and frequently inspected by people. When it is not inspected by people, it is inspected by automated software tools that are developed by people. Software is produced in teams and companies that are comprised of and run by people. The common denominator, i.e., the human element, is critical to advancing software security. When hiring software developers, vendors should perform background checks. Hiring candidates with appropriate security certifications can help assure a certain level of security knowledge and expertise. Once hired, ongoing training of all personnel must be an essential component of any strategy to build secure applications. Developers must be required to know how to write secure code. Quality assurance professionals must be trained to test software for security vulnerabilities. Management needs to understand how to ensure that security is integrated into every step of the development lifecycle. It would be best practice to continually educate everyone involved with product development on common attack patterns and how to protect against them. Some of the well-known sources of common attack patterns that vendors can use are the “OWASP Top 10” (Open Web application Security Project), “SANS/CWE Top 25 most dangerous programming flaws”, the “OSVDB” (which is the Open Source Vulnerability DataBase), and the “CVE” (Common Vulnerability and Exposures) project by MITRE. Armed with these lists, vendors can both focus their educational and remediation efforts to address the most prominent threats. Due to the dynamic nature of the information security field, such lists have to be regularly updated. The OWASP Top 10, for example, is updated every two or three years. Although the list varies with time, many attacks remain on the list year after year.

Forces that Constrain Vendors’ Software Security Practices

**Complexity and Cost**

Considering that even the best security professionals struggle to spot complex coding flaws when reviewing only a few lines of code (Howard and LeBlanc, 2003), the task gets considerably more difficult when millions of lines of code need to be reviewed, redesigned, and fixed. Quality assurance personnel must test the fixes and perform regression tests to ascertain that no product functionality has been lost in the process of correcting flaws.

A development methodology that includes the aforementioned activities, coupled with diligent risk mitigation, inevitably increases the time and resources spent on a project. Experience suggests that there is a direct relationship between the number of features introduced on a project and the number of developers needed. The chance of having a vulnerability introduced to the codebase is directly related to the knowledge and care of the sloppiest developer on a team (Anderson and Moore, 2006): if the number of developers in a team increases, the chance of having a sloppy developer on the team also increases. (Anderson and Moore, 2006) On the other hand, the application security assessment and testing usually depends on the sum of a team’s effort as there are several touch points in the development lifecycle where a group of individuals is responsible to review the work. Putting a premium in hiring fewer, better, and more expensive developers, and, keeping them educated while increasing the number of testers in the organization raises the software production cost and diminishes the number of features added to a new release.

Quality assurance is expensive for producers. It has been estimated that almost 50% of development costs are due to testing (Myers, 1979). While software testing is recommended as a best practice for removing software defects, little is known about how to use these resources devoted to testing in the most cost effective way (Wagner, 2005). A recent study by Zheng, Williams, Nagappan, Snipes, Hudepohl and Vouk (2006) examined whether an organization can economically improve the quality of software products using automated static analysis. Their findings show mixed results. While the cost of automated
static analysis is roughly equivalent to the cost of inspection and testing, the benefits are less conclusive. Zheng et. al. (2006) found no conclusive results regarding an increase in overall product quality as a result of using automated static analysis; and, found that defect removal using automated static analysis was equivalent to results obtained using inspection and testing. Because most software projects are led by companies and companies are profit maximizers (Wagner, 2007), software quality needs to be approached using measures such as Net Present Value (NPV) and Internal Rate of Return (IRR). While it is possible that such measures have been used within a company, these measures have not been widely used in the research literature. Advances in this area could help vendors adopt more efficient and effective testing methods. As testing efficiency improves, more resources can be devoted to secure development and testing.

Other Market Forces

Because customers seem to find value in new features, as evidenced by purchasing trends, software makers have the incentive to create them. Software vendors, like most producers, want to get to market early to capture market share; a delay in the time to market can be costly. While some industry sectors can increase production by increasing personnel, this is not a particularly effective strategy in software development. Having more developers frequently means additional coordination, which often results in time delays (Arora, Caulkins and Telang, 2004).

The “first to market” advantage makes some vendors rush to launch their products at the expense of overall product quality and security. Coupled with the fact that, in the software industry, vendors have the ability to fix bugs later, this leads to a scenario where software vendors launch a product early, then release corrections later as a software update. The study by Arora, Caulkins and Telang (2004) found that patching investments and time of entry to the market are strategic complements. Identifying a bug and writing and testing code to fix it is mostly ‘fixed cost’ in nature, which allows vendors to reasonably build patching into their product costs. A predictable future investment in patching coupled with the need to enter the market early incentivizes vendors to release buggier products earlier and patch them later.

Many customers adopt market dominant applications because of synergy and interoperability created by using the same application others are already using. This phenomenon is known as the bandwagon effect. To foster a larger network of users, vendors create an ecosystem or a platform that appeals to makers of complementary products. Platform adopters desire openness and straightforward interfaces as this represents more product opportunities and lower prototyping costs. More often than not, both attributes are at odds with security. Vendors who spend the time and resources developing a secure architecture for system extensibility may incur the risk of stifling platform innovation as security requirements could get in the way and make life harder for the complementers. Platform vendors tend to ignore security in the first releases until they build a market position (Anderson, 2001). It is generally accepted that security is treated as an add-on. Based on the above discussion, it seems that market forces may be partly responsible for security being considered and add-on.

Unfortunately, most customers lack the ability to distinguish secure from insecure software. A situation where the vendor knows more about the quality of the product than the consumer is considered to be one of asymmetric information. Akerlof (1970), an American Economist, called this the “Market for Lemons”. As a result of insufficient information about product quality, consumers cannot discern what level of quality they are getting for their money. In the absence of information, consumers assume that they will only get average quality and therefore are only willing to pay average prices. The effect over time is that producers will only produce average quality product given that is what sells. Because high quality products are driven out, the market is called “a market for lemons”.

While one might think that the presence of vulnerabilities would suggest product inferiority to customers, research shows that software customers believe that software in a uniquely complex product that is bound
to have defects (Cusumano, 2004). Therefore, it could be that software customers are more willing to accept software defects than would be accepted in other product markets. And, contrary to what one might expect, it is possible that the presence of vulnerabilities actually leads consumers to perceive that the vulnerable software product is superior. Let’s take Microsoft for example. Microsoft is widely known for its vulnerabilities. Many believe that it is Microsoft’s large market share that makes it an appealing target. If hackers want notoriety for their exploits, a large attack target is likely to offer more notoriety. The consequence of this is that it signals product superiority, not inferiority, to customers.

Moving Forward
A possible solution path to the problem of the high cost of creating secure software is to enact legislation that provides incentives to software makers who invest in practices such as training and automation. However, relying on policy to ensure software assurance is not a fool proof solution. Past attempts at standardization, such as the Payment Card Industry Data Security Standard (PCI) made companies race to become compliant but not necessarily secure, diverting corporations to focus solely on compliance. The Common Criteria motivated vendors to shop around for evaluators who would give their products the ‘easiest ride’, for example, by asking fewer questions, charging less money, taking the least time, or all of the above.

Others have suggested that penalties for egregious vendors are needed. For example, taxing vendors who have high vulnerability rates has been suggested. Earlier we noted that many vulnerabilities are known about Microsoft products, in part because Microsoft is an attractive target for hackers. Is Microsoft an egregious vendor or a large target? One challenge with this approach then is in identifying what constitutes egregious behavior. Stiff financial penalties against vendors could also create barriers for newcomers entering the market. With fewer market entrants, the goal of “greater consumer choice” becomes a dream. Homegrown software is developed either by internal developers or by outside contractors, neither of which have the necessary resources to go through the same level of scrutiny that large vendors such as Microsoft can apply to product development. These groups have more insecure development processes than just about any major software vendor. Should financial penalties be levied against vendors, these groups could suffer the most.

Another action that could raise the tide of software security is to grant read access to source code. Advocates assert that this approach will finally treat software as what it truly is – a common good. By allowing read access, consumers and citizens can truly signal what they demand in terms of software security and at what cost. Furthermore, such an approach is in the best interest of society as the technical means of today are too powerful to be in the hands of the few. Advocates note that if citizens are to influence their own future, they must know enough about technology to fulfill their role as citizens; they must be in a position to speak from a position of enlightenment and knowledge regarding technical means. If they are not speaking from this position, they are speaking from a position of ignorance, which is always a position of subservience. Critics note that while the access granted to all consumers and citizens might be the same, the resources for each consumer and citizen to participate will still be asymmetric. Those with resources will be able to participate more fully at the risk of further discrimination to those who cannot. Other critics contend that the motivation to report vulnerabilities will still vary. Suppose a military agency of a government discovers a flaw in a widely used software application. If this flaw is reported to the vendor, who then corrects the flaw, all users (including adversaries) will benefit from improved security. If the agency remains quiet, it could take measures of its own to mitigate the vulnerability, while at the same time, exploit the flaw to attack adversaries.

Summary
Vendors are the creators of the software and have responsibility for creating secure software. Vendors can follow a number of best practices to enhance the security of their code. However, there will never be
perfectly written code. The complexity of code and code development in combination with competing interests among consumer choice, barriers to entry, price, and security continue to challenge progress toward the common good of the development of more secure software. While vendors are creators of code, adopters are also important in the software security equation, which we turn to next.

**ADOPTERS**

Adopters are organizations and individuals who use software. This includes both legal and illegal users (i.e., pirates). For our purposes here, legitimate adopters are organizations and individuals who use software in accordance with current copyright laws. This includes a wide variety of for profit businesses, nonprofit businesses, organizations, associations, government agencies, institutions, and users. These entities range in size and location to include small, locally owned or operated entities (e.g., a local food cooperative, the local school district, and a community credit union) to large national and multinational entities (e.g., the Department of Treasury, Chase Bank, Wal-Mart Stores, Nestle, Royal Dutch Shell and Bayer Group).

Illegal users are those who do not act in accord with current copyright laws when using software. Illegal users range from individual users who share a copy of a piece of software with a friend to organized groups and nation states. Increasingly, groups are organizing and challenging existing copyright practices. Take, for example, the Pirate Party, which was founded in Sweden in 2006. One of the foremost goals of the Pirate Party is to reform copyright law. Specifically the party is seeking (1) modification of copyright law so that all non-commercial copying of software is free, and (2) a complete ban on digital rights management technologies (The Pirate Party, 2009). On a larger scale, the World Trade Organization (WTO) was established in 1995 to deal with rules of trade between nations. The WTO has outlined rules on the intellectual property protection and enforcement for the multilateral trading system (World Trade Organization, 2009). However, several countries where software piracy is rampant are not members of the WTO and do not acknowledge these rules. And in some cases, countries that are members still fail to abide by the agreement.

The relative recency of the WTO ruling and the development of the Pirate Party show that issues of software ownership are anything but resolved. As these issues play out, it makes the contributions of the myriad of adopters to the state of secure software use anything but clear. There are multiple competing interests, and ineffective and incomplete practices that merit a closer look, not so much for providing a full characterization, but to at least approximate some of the systemic challenges.

**Adopter Practices Regarding Software Security**

Across organizations of all sizes, type, and locale, one trend seems clear - the responsibility for information security within organizations is evolving. In the past, information security was relegated to the information technology department and primarily viewed as a technical and system concern. Today, information security is increasingly being regarded as a matter of governance within organizations where the focus is on strategic alignment of information security to business objectives and value delivery, as well as attention to risk, resource, and performance management. According to the IT Governance Institute (2006), information technology governance has the following goals:

- Increase share value for organizations that practice good governance
- Increase predictability and reduced uncertainty of business operations by lowering information security-related risks to definable and acceptable levels
- Reduce potential for civil or legal liability as a result of information inaccuracy or the absence of due care
- Enhance the structure and framework to optimize allocation of limited security resources
- Improve assurance of effective information security policy and policy compliance
• Fortify the foundation for efficient and effective risk management, process improvement, and timely incident response related to securing information
• Increase the level of assurance that critical decisions are not based on faulty information
• Increase accountability for safeguarding information during critical business activities, such as mergers and acquisitions, business process recovery, and regulatory response (pg. 13).

As information technology governance evolves, one best practice is for organizations to adopt a cost benefit model to manage the return of their security investment. Costs are the programs and initiatives instituted to reduce risk. Costs can be categorized in four broad categories: transitory, long term, tangible, and intangible (Cavusoglu, Mishra, and Raghunathan, 2004). Transitory costs include things such as lost business and decreased productivity as a result of system downtime, as well as costs to detect, contain, repair, and restore the system, and costs to prosecute and notify customers and the public. In contrast are long term costs such as customer loss (both customers who leave as a result of the breach, as well as the loss of potential new customers), potential increases in insurance, and higher capital costs in debt and equity markets. Costs such as lost sales, material and increased insurance are calculable and therefore tangible, while costs associated with decreased trust, such as loss of potential new customers, are more intangible (Cavusoglu et. Al., 2004).

Using a cost benefit approach, reduction of information security risk can be treated as the intended benefit. Organizations that adopt a cost benefit approach will need to think purposefully about how to measure the benefit. Research has shown a relationship between the announcement of information security breaches and loss of market value. Ranges on market devaluation following an information security breach have been reported from 2.1 percent of market value within two days of announcement (Cavusoglu, Mishra, and Raghunathan, 2004) to 5.6 percent over a three day period following breach disclosure (Garg, Curtis, and Halper, 2003). An expected benefit of reducing information security risk would be to preempt the possibility for this type of market devaluation. Another way for assessing information security risk would be the Risk Management Guide for Information Technology Systems published by the National Institute for Standards and Technology (Stonebruner, Goguen and Feringa, 2002). By using this framework to measure information security risk, adopters can develop a risk baseline. Then, as security investments are made, risk can be assessed again. Reductions in risk can serve an indicator of benefits of the program. While business analysts can clearly articulate the need for a cost benefit approach to software security, operationalizing such studies is still a challenge due to the interdependent nature of risk in this area as mentioned earlier.

Errors in system administration, configuration, and maintenance are often reported by the media as causes of security breaches in large enterprises (Greenemeier, 2007). Hence, there are a variety of adopter best practices regarding system administration, configuration, and maintenance. Responsible use of software starts with choosing the right solution. Due to the high incidence of new security attacks, patching becomes an essential ongoing activity to keep the software secure. Patching needs to be augmented with other key operational tasks. For example, no system is secure without addressing physical security. Large institutions often run internal pilot tests of products before deployment. The goal of pilot programs is to make sure the software runs as expected in the company’s unique environment and that no vulnerabilities are introduced to this environment. Software must be configured properly. Applications should encrypt sensitive data in storage and in transit. The use of firewalls, antivirus software, and intrusion detection is essential to provide defense in depth. Strong physical security at data centers is critical. Systems entrusted with sensitive data must be isolated both physically, using different hardware, and logically, using isolated networks and access controls. Whenever possible, a company’s financial and HR system must not be directly accessible by external networks such as the internet.

Patch management is an important part of every IT administrator’s responsibilities. By ensuring that the latest security patches are installed, the company mitigates the risk of exploitation of vulnerabilities that
 Software security patches must be up-to-date for all adopted software; and, all unnecessary system functionality needs to be removed. Restricting user access and limiting accessibility to applications reduces the attack surface and therefore, the vulnerability (Howard and Lipner, 2006). Access control must be carefully planned to ensure that users do not receive more privileges than necessary to perform their work. Access to servers must be logged on a separate machine for a potential audit. Strong password policies must be enforced to mitigate brute force attacks, including frequent password change and prevention of password reuse.

Forces that Constrain Adopters' Software Security Practices

Operational Challenges

Clearly, there are always risks associated with choosing products. In the case of an enterprise, the team entrusted with this responsibility must be held accountable for selecting the products that best fit a company’s needs. Too often, business motives conflict with security concerns. For example, the people testing new products or the individuals responsible for the decisions are not the same individuals who will support and pay for the consequences when the product fails. Incentives for the ultimate decision of purchasing a product over another may be related to a generous discount offered to customers who buy a product bundle with another product offering, extended or premium support, or a strategic business partnership. Managers who opt to go with well known brand names often don’t get fired, even if the product proved to be inadequate as they chose the market leader. Companies may allow security maintenance to take a back seat during critical periods such as end of quarter deadlines.

Despite vendor’s best efforts and guidance at installation, complex systems often communicate with other in-house applications that may not adhere to the same security standards. This potentially poisons the system with tainted data or forces companies to use the lowest common denominator to exchange data, which is often insecure.

The trend of outsourcing IT infrastructure and services goes against the principles of having strict control over data access. Although outsourcing may bring higher profits and convenience to an enterprise, it often results in less data security.

The Challenges of Patching

There are numerous challenges associated with patching that hamper adopters’ security practices. Unfortunately, due to the urgency of some patches, administrators don’t have much time for informed decision making or patch testing. Most major attacks tend to occur within hours of the release of a security patch. Upon release of a security patch, attackers reverse engineer the code, identify the vulnerability and subsequently develop and release exploits, hitting organizations before they are protected. Administrators need to act fast to safeguard their networks, while keeping in mind that deploying a faulty patch can result in significant downtime. A common pattern observed in the industry is worm creation following the patch availability. If a company is victim of a worm, it is likely that the business failed to exercise their duties properly to maintain their environment and thus protect customer’s data.
Security patches are released frequently, so effective patching requires an ongoing commitment to timely response. Users may not know how to patch their systems, especially in the case of home users or smaller businesses. Applying a patch takes time and can sometimes break a system, rendering it unavailable for a period of time. While diligent patching is considered a best practice for software adopters, the costs associated with updating systems and the accompanying unavailability of systems forces IT to only provide maintenance at certain windows of time. Large companies usually don’t allow system updates to be performed during periods such as the end of the quarter due to the potential impact on revenue. Unpatched systems increase the attack surface, making every adopter, to some degree, reliant on the patching practices of others. Add to this that pirated software is almost always unpatched and the attack surface is larger.

The ability of adopters to patch is affected by patch release policies and practices that vendors institute. The question of when patches should be released is contentious. Some believe that releasing patches on a Friday is a poor choice since most adopters will not be able to deploy the fix promptly. Attackers, on the other hand, will work during the weekend to exploit vulnerabilities on unpatched computers. Vendors must also be sensitive to other countries’ holidays and festivities. Microsoft releases security patches during the second Tuesday of the month. This date, also known as “Patch Tuesday,” was chosen to avoid the beginning of the week (which is still the weekend in some time zones), but also to be far enough from the end of the week, allowing time for companies to resolve any problems that may arise due to the introduction of changes to the environment. While this allows system administrators to coordinate updates and plan accordingly, it also allows criminals to schedule their attack efforts. The term “Exploit Wednesday” has been coined by the media to describe all new assaults following Microsoft’s update announcements. Patch coordination can be convenient to customers, but it also poses a danger by defining a window of exposure before the patch is released giving criminals a schedule for working on attacks.

Microsoft’s Patch Tuesday has created unexpected difficulties for other software vendors. No other large software vendor has been able to conduct a similar patch release on the same day. Due to Microsoft’s large market share, the number of important updates on the second Tuesday of the month can be overwhelming and often receives extensive media coverage. Vendors who have tried to schedule their own patch releases for the same date to simplify customers patch procedure have been accused of “hiding behind Microsoft” by releasing updates unnoticed by the media. Overwhelmed system administrators have also objected to this practice leaving the remaining vendors only with Wednesdays and Thursdays and the remaining Tuesdays of each month to release new patches.

There are also quality issues associated with patches. Users may not patch their systems immediately either because they find them too difficult to patch or the quality of the patch is not consistent enough that people can feel safe patching right away. From the vendor’s perspective, if there is slow uptake of the patches by the end users, the pressure on vendors to develop the patch in less time is reduced. It might seem that if the vendors are given more time to patch for the vulnerabilities, then the quality of the patch will be better. Unfortunately, research shows that there has been only marginal benefit in the quality of the patches (August and Tunca, 2008).

As we know, the software industry is profit maximizing; a factor that some people believe creates dis-incentives for vendors to release patches. A case in point is the “Genuine Advantage” program launched by Microsoft in 2005. Genuine Advantage allowed users to download service packs on the condition of authentication of their Windows XP operating system. In essence, Genuine Advantage denied pirates the opportunity to patch. Critics contend that Microsoft was differentiating legitimate users from illegitimate users in hopes of converting some of the latter, thereby increasing revenue and protecting shareholder value. The unintended consequence was an increase in the number of unpatched hosts (an increase in the attack surface), exacerbating the security problem. The concern is that in
protecting itself and its legal customers, Microsoft was in fact attempting to maximize its own welfare (profit) at the expense of all. This case demonstrates how patching, piracy, and vendor practices are entangled; piracy affects patching, and both are shaped by a variety of factors including market economics, local cultural norms and the diverse intellectual property policies of multiple societies (Business Software Alliance-International Data Corporation, 2008). Here we refer readers back to chapter one where Harter (quoting Ackoff, 1981) defined a mess as a “set of two or more interdependent problems (p. 52).” It is in this example that we clearly see how economic systems are interacting with technical systems and human systems in a way to produce emergent properties…we see “the mess”.

**Moving Forward**

Toward the beginning of the chapter we discussed the interdependent nature of information systems and therefore the interdependent nature of information security risk. While adopting a cost benefit model is viewed as a best security practice, it is important to keep in mind that accurately measuring information security risk is quite difficult, as we discussed earlier. Therefore, determining what efforts should be made to invest in information security and predicating investments on expected benefits are considerable challenges.

In the section immediately before this one we alluded to the interdependence between vendors’ patch decisions and adopters’ patch practices on the state of security. We also noted that piracy, patching, market economics, local cultural norms, and intellectual property policy are intertwined issues. The field is beginning to see research studies that investigate the nature and context of this interdependence.

A study by Rahman and Kannan (2007) investigated whether vendors who restrict patches only to licensed users complement or substitute government efforts to enforce anti-piracy with respect to social welfare. They found that when anti-piracy efforts are high (when there are high levels of public policy) and the cost of developing a quality patch is high, the vendor does not benefit from restricting patches. In other words, once a certain level of government intervention is provided, it optimizes both vendor profit and social welfare to distribute the patch universally (to both legitimate and illegitimate users). A related study (August and Tunca, 2008) investigated the conditions under which a policy that allows only legitimate users to patch is optimal for the software vendor and social welfare. The conditions allowed for in this study were security risk, piracy enforcement, and piracy tendency. Their findings suggest that it is in vendor’s interests to restrict patching to licensed users under the following conditions: (1) high security risk combined with low piracy enforcement, or (2) low piracy tendencies in the consumer population combined with high piracy enforcement levels. Regarding social welfare, the findings of August and Tunca (2008), which are counter to that of Rahman and Kannan (2007), suggest that when piracy enforcement is high, restricting patches to licensed users can be socially optimal. Despite these differences, what these studies reveal is the complexity of formulating optimal patching and piracy policy for users and vendors by accounting for software product characteristics, consumer market conditions, local cultural norms, and intellectual property policy. More work in this area is needed.

A loss of market value is a loss of economic welfare for the company and its shareholders. What is less clear is the potential impact of the vulnerability on the state of security and social welfare. The challenges in estimating costs and benefits (risk reduction) are not just important in order for organizations to make investment decisions, such measures are important for informing discussions of the overall state of security as a matter of social welfare and public policy.

**Summary**

From the discussion above, we can see that sound information security practices offer numerous benefits for adopters seeking to reduce software security harm and risk. While investing in good information security techniques can provide adopters an advantage, improper implementation of such practices
remains an issue. Furthermore, due to highly interdependent nature of information systems, the implementation of such practices cannot address all of the harm and risk posed by insecure software. Despite best efforts to build, deploy and govern secure software, some portion of software vulnerabilities is inevitable, which brings us to the role of vulnerability disclosure.

**VULNERABILITY Disclosure**

Vulnerability disclosure refers to the publication of information about a security problem. Questions about vulnerability disclosure include when, how, what, and to whom vulnerabilities should be disclosed. Vulnerabilities may be reported by full disclosure or by responsible disclosure. Full disclosure refers to a situation where information about how to detect and exploit the vulnerability is posted on public websites after discovery. In contrast, responsible disclosure describes the situation where vulnerability is first disclosed privately to a vendor, and the finder works jointly with the vendor to solve the problem. The vulnerability is made public only when a patch is available.

**Full Disclosure**

Full disclosure practitioners believe that publishing vulnerability information immediately after the vulnerability is discovered is desirable for a variety of reasons. Some full disclosure advocates support this practice on the premise that enhancing user and public awareness is critical as user action (or inaction) is an undeniable part of the security equation. By disclosing fully and immediately, users will patch systems thereby making us all more secure. A second rationale for full disclosure is that fixes will be produced faster because vendors are pressed to respond in order to protect their reputation and market share. A third justification for full disclosure is that the free flow of information may help other vendors to provide attack prevention solutions such as firewalls, antivirus software, and intrusion detection systems (IDS), which could take less time to produce compared to fixing fundamental architectural flaws in the vulnerable software.

Today’s practice is that once the information about vulnerability is public, it is the vendor’s responsibility to either confirm or deny the information. As the definitive authority on the product, this affirmation typically includes a rating of the severity of the vulnerability, a fix schedule, and a statement about the vulnerability of previous versions and similar products. Under the full disclosure model, the vulnerability is announced prior to a patch being available, therefore, whenever possible, the vendor should suggest a workaround for the problem, such as disabling the flawed functionality or blocking communication ports until a patch is available.

Some full disclosure advocates favor immediate release based on reducing the window of exposure. Others contend that it is irresponsible to disclose, discuss, or confirm security issues until a full investigation has occurred and any necessary patches or releases are available. The rationale for keeping the vulnerability secret is that immediate full disclosure provides detailed information to attackers before a defense mechanism is available for users. Some contend that attackers are able to develop better exploits and share those exploits among the attacker community, thereby increasing the potential for strong attacks. While it is possible to quantify the number of days it takes for vendors to fix a certain vulnerability that was fully disclosed, it is almost impossible to quantify the number of attacks that succeed due to the knowledge about unveiled breaches.

Given the potential for harm, vendors have attempted to sue finders who practice full disclosure. Alex Halderman, a Princeton PhD student, was threatened by SunnComm with a ten million dollar lawsuit for exposing a weakness in the Media Max CD3 product that allows a user to duplicate copyrighted material (Smith, 2003). SunnComm maintained that the disclosure affected the company’s reputation and caused its market value to drop by more than $10 million. In this particular case, the student did not reverse engineer the product, but only used a well-known and documented OS feature to achieve the reported
result. In essence, he described how the normal use of the operating system could cause undesirable results in the Media Max CD3 application. SunnComm eventually reversed the decision to sue the graduate student. SunnComm CEO acknowledged his threat to file a lawsuit was a mistake and that “the long-term nature of the lawsuit and the emotional result of the lawsuit would obscure the issue, and it would develop a life of its own” (McCullagh, 2003).

Hostile vendor response to full disclosure can hinder hardening of products and can strain the relationship between software vendors and adopters. If vendors should bear responsibility for the security of their products, one wonders if rather than threatening Mr. Halderman with a lawsuit, the vendor should somehow reward him for going above and beyond the company’s best efforts to find flaws during the test phase of the lifecycle and contributing to the enhancement of its product line? Was SunnComm’s stiff reaction due to the harm caused by the flaw or due to pressure from the makers of copyrighted material?

Responsible Disclosure

Responsible disclosure intends to ameliorate some of the concerns raised regarding full disclosure. Under responsible disclosure, practitioners (or “finders”) notify the vendor first to allow a reasonable timeframe to fix a problem. Once a fix is released, a finder may or may not publish full details about the vulnerability. Although vendors prefer not to give financial rewards to finders, it is common industry practice to publicly credit them for their work when a fix becomes available. Responsible disclosure aims to allow a vendor enough time to apply best engineering practices. These best practices improve both the quality of the fix and diminish the chance of introducing new flaws in the product. Solutions can be back ported to previous versions and fixes can be made available on internationalized versions of the product by the time the patch is announced.

While best engineering practices are desirable, they can significantly increase the time needed to produce a fix, thus increasing the window of exposure. Enterprise customers may not welcome frequent patches and do not always deploy them quickly. They often prefer fewer releases due to the cost of deployment and prefer installing a single patch that resolves several vulnerabilities (Viega, 2009).

Although many vendors are committed to excellence, others may delay security fixes in favor of revenue-generating feature enhancements and bug fixes. When faced with unacceptable delays, the finder is left with the dilemma of either trusting that the vendor is responding reasonably and acting in good faith, or demanding a shorter timeframe on behalf of the user community. For the finder, the ultimate weapon against negligent vendors is to threaten them with a full disclosure.

Once a fix is released, some finders publicize details of their findings. Unfortunately, the same contributions that ought to lead to improvements in security can be used to cause harm. According to Viega (2009), over 95% of the malware that leverages security flaws uses vulnerabilities whose details were published on the internet. The role of Good Samaritan and user advocate is suddenly challenged by the perception that finders are after self-promotion and financial gains from selling security assessments. According to this train of thought, it is not in the economic interests of finders to put off taking credit for finding vulnerabilities, even though users may be hurt.

Full vs. Responsible Disclosure: More to the Story

Clearly, the debate of full vs. responsible disclosure is current and multi-dimensional. According to Schneier (2007), responsible disclosure, by definition, requires secrecy, which in turn prohibits public debate about security. Inhibiting the free flow of information hurts security education, and security education leads to improvements. Information secrecy prevents citizens from accurately assessing their own risk and from making informed decisions about security. Other experts argue that when systems
carry life-threatening flaws (such as a defect in an airport control system that could lead to an airplane crash,) public awareness is a necessity regardless of whether or not a fix is currently available.

Given the debate over which is the better approach to vulnerability disclosure, Cavusoglu, Cavusoglu and Raghunathan (2004) investigated how vulnerabilities should be disclosed in order to minimize the social loss. In this study, social loss was defined as the vendor’s patch development cost and the damage and workaround costs incurred by adopters. The study looked at three disclosure models: full vendor disclosure, immediate public disclosure, and a hybrid approach. They found that none of the disclosure models is always optimal. Rather, the findings of this study suggest that the optimal approach to vulnerability disclosure is stochastic and the main determinants are the characteristics of the vulnerability (i.e., risk before and after disclosure), the cost structure of the software user population, and the vendor’s incentives to develop a patch.

Arora, Telang and Xu (2008) examined how vulnerability disclosure policy can optimally balance the need to protect users while providing vendors with incentives to develop patches expeditiously. Their model suggests that the optimal disclosure policy depends upon the behavior of vendors, potential attackers, and users. When vendors do not internalize the entire user loss, they will release the patch later than what is in the best interest of users, unless they are threatened with disclosure.

Arora, Nandkumar and Telang (2006) investigated how attack propensity changes with the disclosure and patching of vulnerabilities. In contrast to the Cavusoglu et. al. (2004) study, this research sought to identify which policy (full instant disclosure regardless of patch availability vs. limited or no disclosure) is optimal based on reducing attack frequency over time. Findings suggest that patches do, in fact, provide crucial information to attackers, underscoring the need to think carefully about efficient and effective means for managing patch dissemination.

Studies such as these have the potential to provide important insight into the nuances of when, how, what, and to whom vulnerability information should be reported. As we can see from these studies, the real questions are not about full vs. responsible disclosure, but the conditions under which a particular disclosure policy may be better than another.

**Approaches to Fix Flaws Discovered In-house**

During the normal process of fixing defects and developing new application releases, software manufacturers occasionally find security flaws in their own products. Two common approaches for correcting flaws discovered in house are silent patching and responsible disclosure. Companies who follow the silent fix process resolve vulnerabilities in the product without public disclosure. Fixes are shipped with new releases of the product. Google follow this process to patch weaknesses on the Chrome web browser (Duebendorfer and Frei, 2009).

Vendors who implement silent fixes assert that the practice increases the difficulty of attacks as criminals are forced to apply sophisticated reverse engineering techniques to uncover details about the flaws. In the field, this argument is countered by noting that hackers are proficient in the art of reverse engineering. According to Duebendorfer and Frei (2009), silent updates force immediate updates for all users and therefore achieve the highest number of users with the latest patches installed. While conclusions are drawn from observing the percentage of updated browsers accessing Google over a period of time, the study makes no mention about how often the updates fail and break the browser. Due to the unavailability of information, silent fixes can hurt customers who depend on information (such as the severity of vulnerabilities) from software manufacturers to determine when to deploy patches and new releases.
The responsible disclosure process for internally discovered vulnerabilities is analogous to responsible disclosure of vulnerabilities found by outside researchers and customers. The only difference is that no one is publicly credited with the finding as companies consider it part of their job. It is believed that most large software vendors opt for responsible disclosure. Once again though, it is difficult to define a “reasonable timeframe to fix the problem”. Given that no one but the software manufacturer knows of this vulnerability, the decision on timing rests solely with the software manufacturer.

Some believe that vendors who follow a responsible disclosure process tend to back port the fixes of internally discovered vulnerabilities to earlier product versions, while vendors who tend to use the silent fix process fix only the latest versions of the product. Practitioners of the silent fix philosophy argue that the process avoids costly coordination efforts related to responsible disclosure. They also assert that silent patching provides near instant updates and gives all users the latest version of products, which helps to cut the cost of customer service and product support, thereby enhancing social welfare. Critics of silent patching claim that the silent fix approach takes away customers’ control of their environment. Patches can cause applications to malfunction. Without knowledge of updates, customers cannot evaluate how updates will affect their environment.

The Market for Vulnerabilities
On the premise that meaningful, high quality vulnerability research should be rewarded and that customers have a right to know, firsthand, about vulnerabilities that may impact revenues, companies such as iDefense and TippingPoint have emerged. These companies purchase vulnerabilities from finders. Once a vulnerability is tested and confirmed by iDefense, they notify their subscribers and appropriate software manufacturers about the finding. iDefense does not go public with the information and neither do any of the customers. Instead, iDefense assumes the finder’s duty of communicating with the vendor and providing sufficient information to create the fix. Once the fix is out, iDefense pays the finder for the research. Some note that this phenomenon, the creation of a market to profit from software flaws, seems ethically questionable. If the normative ethical position is that moral agents ought to do what is in their own self-interest, developers working for software vendors may be enticed to purposely create flaws in their code and later sell them. Companies like iDefense and TippingPoint could inflate the market value of their subscription service by leaking vulnerability information to harm nonsubscribers. In addition, the motives behind customers who are buying these types of services are unknown; and, they may intend to profit from the flaw rather than to use the information to improve their security. Furthermore, introducing a “middle man” could delay problem resolution. On several occasions in the past, it took over one year for iDefense to report vulnerabilities to software manufacturers (Elf, 2004).

Summary
There are pros and cons associated with each of the approaches for the vulnerability disclosure as evident from the discussion above. However, it seems that the more we know about optimal vulnerability disclosure, the more we know what we don’t know. What the chapter, thus far, shows us is that responsibility for harm and risk of software security flaws is a complex process with numerous parties, and various competing interests – i.e., it is a mess. We return now to our discussion of the common good. We outline key issues that especially challenge the effort to attaining the common good of software assurance. And finally, with the idea that “the law should conform to ethics, not the other way around” (Stallman, 1992, p. 1), we introduce the role of government with regard to responsibility for harm and risk of software security flaws.

THE COMMON GOOD: THE ROLE AND CHALLENGES OF GOVERNMENT INTERVENTION
In ethics, maximization or optimization is the concept of always doing the act that yields the greatest return (“Maximization”, 2008); this is the ideal of the common good. Given that it is the polis that bears responsibility for the common good, the role and challenges of government intervention are timely and relevant.

One role for government is to enact public policy, laws, and regulations that seek to advance the common good and enhance social welfare. Laws and regulations are essentially rules that aim to induce people to do what we want them to do. Inducements can be positive or negative. Positive inducements, also called incentives, seek to make it easier and more rewarding for people to do what we want them to do. Deterrents, or negative inducements, seek to make it harder, or more costly, for people to do what we do not want them to do. We turn to a discussion of public policy and its role in responsibility for harm and risk of software security flaws.

**Profit, Welfare, Laws and Software Security**

As we know, the actions that each user, vendor, and adopter takes is influenced by multiple incentives and disincentives and produces several consequences for other users. Software vendors as profit maximizers patch their systems not because of social responsibility but rather because of the profit motive, i.e., the market measures and rewards companies by how much profit they can generate. Patching is only incidental to this goal.

For users, patching becomes a matter first of knowledgeability, and assuming full knowledge of the need to patch. Patching for users becomes a matter of how frustrated they are by unpatched software compared to their frustration with having to test and deploy yet another patch. Assuming the user frustration with unpatched software is greater; a vendor can decide whether the cost of possibly losing these users is worth the cost of creating the patch and how to create patch deployment that decrease vendors’ cost of patch efforts. Today, unregistered copies of Windows are not entitled to receive security patches. For this reason, most pirated copies of Windows are infected with viruses. This phenomenon threatens the security of all legal users, and Microsoft’s reputation as a secure vendor. If sales decrease due to a perception that Windows is more prone to virus infections than its competitors, the company would be financially better off providing free security updates to all users, even those using pirated copies of windows.

A key question is how to balance the point of view of the profit maximizing vendor with the needs of adopters and users in a manner that is welfare maximizing. August and Tunca (2006) compared the status quo of consumer self-patching effects with three patching options: (1) mandatory patching, (2) patching rebates, and (3) usage taxes to determine the incentives each option and effects regarding profit maximization and welfare maximization. Further, they compared these patching options for proprietary versus open source software. It was observed that mandatory patching is not useful in the case of open source. In the case of proprietary software, contractually mandating consumers (option 1) to patch does not improve vendor profit and is usually not helpful in increasing the social welfare (August and Tunca, 2006). The primary reason for the ineffectiveness of mandatory patching is that the consumers are forced to commit to the potential costs when they purchase the software, which negatively influences their purchasing behavior. This observation suggests that the patching decision should be left to the consumers and other ways to improve the users patching behavior should be investigated.

Option 2 is to provide users with increased incentives to patch by offering rebates to patching customers. According to August and Tunca (2006), there are two way for determining the amount of rebate to be given to its customers. The first is for the rebate amount to be determined by the vendor and the second is for the rebate amount to be determined by a social planner (a decision maker who attempts to achieve a result that is in the best interest of all parties). In the case of vendor-determined rebates, August and Tunca (2006) found that when both the patching cost and the effective security risk are high, the vendor...
must price low to induce purchases. In such cases, by offering rebates, vendors can induce an increased patching population and increase the security of the product. On the other hand, when the expected security risk is low as compared to the patching costs, it becomes expensive for the vendor to incentivize consumers to patch and rebates can result in losses for the vendor. In the case of social planner-determined rebates, when the security risk is high and patching costs are high, under vendor’s optimal pricing, the patching population is small. Therefore, forcing the vendor to assume part of the risk by paying a rebate to the patching consumers may increase social welfare. On the opposite side, if the cost of patching is low, forcing the vendor to offer a rebate can decrease the social welfare by inducing inefficient patching behavior.

Given that poor patching behavior by the users introduces security risks on the entire user population, August and Tunca (2006) also investigated whether a usage tax would drive a certain group of users of the usage pool, thereby increasing overall security. They found that imposing a tax decreases the vendor’s optimal price, but the price plus the tax, i.e., the effective amount that the consumers have to pay to use the software is larger than the optimal vendor price with no tax. Therefore, they concluded that taxes neither increase vendor profits, nor increases social welfare for proprietary software. Interestingly, a usage tax is the best policy for open sources software except when costs and risk are low, where a rebate policy still prevails. While the findings of this study should be of interest to vendors, adopters, and policy makers, more research is needed.

Furthermore, we have to remember that software security is entangled with other issues of information security. “Hacking tools” such as Metasploit, Ice Pack, and L0phtCrack significantly lower the technical knowledge necessary for attackers to commit transgressions against software. With the aim of minimizing computer attacks, an amendment to the Computer Misuse Act was proposed in the United Kingdom to make it illegal to supply (i.e., create and distribute) software that is likely to be used to commit an offense. Therefore, policy analysis needs to look at the effects of policies that ban tools used by hackers, the reduction on cybercrime, and consequent effects on the state of software security.

Governments have also increased the enforcements associated with different computer related violations. Take for example, the punishments for the following computer intrusions. Samy Kamkar, the author of the MySpace worm (Kamkar, 2007) was sentenced to 3 years probation, 90 days of community service, and an undisclosed amount of restitutions. Computer hackers, Eric McCarty, breached the University of California’s (USC) registration website (Lemos, 2006). McCarty reported the vulnerability to USC and to SecurityFocus. This unauthorized intrusion resulted in a penalty of probations, home detention, and restitution for McCarty. Policy analysis needs to look at the effects of enforcement policies on deterrence, and the consequent effects on the state of software security.

**Challenges of Policy Solutions**

The measure of an effective public policy is its ability to address a problem. By nature, public policies target a specific locus of control to serve as leverage to fix the problem. Applying public policy to software security will be a challenge because the responsibility for harm and risk of software security is a complex causal situation. Both the causes and the effects of software vulnerabilities are diffuse and many. In situations where causes and effects of social problems are diffuse and rooted in a complex organizational system, inducements that are narrowly applied, as is the case with most public policy, are unlikely to have significant impact. It does not mean that public policy will not be enacted. It will. What happens though, is that that narrowly created policy is enacted over time, gradually addressing portions, hopefully the most significant, of the problem while leaving other portions unaddressed.

It also happens that inducements have unintended consequences, sometimes hurting the very things one is trying to protect. An example of a law that has had unexpected side effects is the German “anti-hacker
"law", enacted in June 2007, to ban hacking tools from its territory (Blau, 2007). As a result of this law, several organizations moved overseas to host the forbidden content and launch attacks from outside Germany. The law made no exceptions for security professionals who perform security assessments, thus limiting the effectiveness of these firms. Another example is the Eric McCarty case mentioned earlier. McCarty accessed only 7 USC database records and reported the vulnerability to SecurityFocus.com. SecurityFocus.com, in turn, notified USC about the flaw through a responsible disclosure process (Lemos, 2006). Despite this fact, USC was obligated by law to notify all 275,000 individuals in the database and shut down their registration service for 2 weeks (Yang, 2006). The estimated cost to USC was $140,000, which became the value of the law suit against the 25 years old would-be-USC student. Clearly, McCarty broke the law, but were his acts malicious or unethical? Will punishments like this genuinely help to ensure society’s safety and peacefulness in the future?

Other Possible Government Roles

Another point of leverage the U.S. government has is its buying power. The Federal Desktop Core Configuration (FDCC) mandate is an example of the use of leverage. The FDCC is an OMB (U.S. Office of Management and Budget) mandate implemented in February 2008. It requires all Federal Agencies to standardize the configuration of approximately 300 settings on each of their Windows XP and Vista computers. The reason for this standardization is to strengthen Federal IT security by reducing opportunities for hackers to access and exploit government computer systems. As a result, several large vendors such as Microsoft and Symantec had to adhere to a list of configuration settings considered secure. It is generally believed that employing this standard yielded considerable government savings. Numerous other security focused organizations have also benefited from this mandate by adopting FDCC in their own environments.

Concerns with vendors’ security practices made the Government explore in new mandates. Based on lists of well known and well understood attack techniques, the state of New York created an application security procurement language document which attempts to make software makers liable for their work. The goal is to give consumers a means to fight back when a minimum standard of due care is not achieved. The effectiveness of this mandate remains to be seen.

Vulnerabilities in widely used open source components and protocols affect a large number of users and demand coordination with multiple parties. Another role for government in this emerging landscape may be as coordinator and mediator of vulnerability disclosures in such cases, as it did in the past with the Sendmail vulnerability (Palella, 2003).

The fast paced dynamics of the field may make some laws and government efforts obsolete. While the laws in some countries traditionally have approached software as a product much like a shirt or a book, the entire concept of software has evolved into something more like a service. In addition, the infancy of the software assurance field imposes a real challenge for governments to determine the right level of incentives to provide.

CONCLUSIONS

As we have seen from the discussion above, the different parties involved have their own set of responsibilities, weaknesses, and interests. On the one hand are vendors, who are motivated to maximize their profit at the expense of the degree of security built into their products; with a primary objective being to maximize their market share and gain competitive advantage by bringing new products into the market in each business cycle. Factors such as complexity and cost to deliver more secure products conflict with market forces, yet this state of conflict does not absolve vendors of some responsibility for harm and risk of software flaws. For making vendors more liable for their products, a number of suggestions have been put forward, such as giving incentives to those organizations that invest in good
programming practices, handing penalties and taxes to those who have high vulnerability rates, and granting access to source code to enhance the security of the code.

Adopters, on the other hand, have their own share of issues to deal with in implementing security practices within their organizations. Although the incentives to invest in security are many, the real difficulty lies in its implementation. Security isn’t a technical problem only, but it encompasses other aspects as well, incorrect configurations of the systems, insufficient firewalls rules, inept policies, patching issues, low levels of security culture among the staff are some of the many reasons behind the operational failures.

The hard part of the problem of responsibility for harm and risk of software flaws, though, stems from interdependent risk, as discussed at the onset of the chapter. Given the diffuse and polycentric nature of this problem, vulnerability disclosure policies and practices have become important mechanisms for attempting to “raise the tide”. The two models for vulnerability disclosure are full disclosure and responsible disclosure. While pros and cons of full and responsible disclosure models have been identified, more research is needed to improve our understanding of the conditions under which different vulnerability disclosure policies are effective.

We have also seen that government is responsible for enacting public policy, laws, and regulations that seek to advance the common good and social welfare. One challenge is in striking an effective balance among the different parties responsible for harm and risk of software security flaws: vendors, adopters and users. Even then, there are unintended consequences, hurting the very things one is trying to protect. The interdependent nature of the risk makes it difficult to develop policies that address causes of risk. Interdependent risk implicates interdependent responsibility. The logic of collective action (Olson, 1971) is such that in small groups, where each member gets a substantial proportion of the total gain simply because there are few others in the group, a collective good can often be provided by the voluntary, self-interested action of the members of the group. However, in the case of software security, the existence of a large group is obvious. While we do not have clear answers, we believe the real challenge is in finding the point at which the benefit to the group from having the collective good exceeds the total cost by more than it exceeds the gain to one or more groups. Only then we can achieve the objective of welfare economic and common good to help society make better choices with regard to the use of the available resources.

ACKNOWLEDGEMENTS
We got useful comments on drafts of some of this material from Wesley Higaki, Robert Hoffman, Shelley Mahr and Jessica Johannes.

REFERENCES


ADDITIONAL READING


---

1 Regarding pareto improvements, it should be noted that there can be numerous pareto optimal alternatives. Pareto measures are descriptive theoretical measures, they describe what is or could be – and they do not describe what should be. The role of welfare economics is in making these options manifest and perceivable. The subjective evaluation of which option to choose given consideration of liberty, equity, justice and so on, is the role of the policy maker.