Software, Vendors and Reputation: An Analysis of the Dilemma in Creating Secure Software

Conference Paper *in* Lecture Notes in Computer Science · December 2010

DOI: 10.1007/978-3-642-25283-9_23 · Source: DBLP

Craig Wright
Association for Computing Machinery
27 PUBLICATIONS **453** CITATIONS

Software, Vendors and Reputation: an analysis of the dilemma in creating secure software.

Craig S Wright
School of Computing and Mathematics
Charles Sturt University
Wagga Wagga, NSW, Australia
craig.wright@information-defense.com

Abstract: Market models for software vulnerabilities have been disparaged in the past citing how these do little to lower the risk of insecure software. This leads to the common call for yet more legislation against vendors and other producers in order to lower the risk of insecure software. We argue that the call for nationalized intervention does not decrease risk, but rather the user of software has an economic choice in selecting features over security. In this paper, we investigate the economic impact of various decisions as a means of determining the optimal distribution of costs and liability when applied to information security and in particular when assigning costs in software engineering. The users of a software product act rationally when weighing software risks and costs. The choice of delivering features and averting risk is not an option demanded by the end user. After all, it is of little value to increase the cost per unit of software if this means that users purchase the alternative product with more features. We argue that the market models proposed are flawed and not the concept of a market itself.

Keywords: Security, Derivatives, Vulnerability Market, Software Development, Game theory

Glossary of Terms:

SDLC	Software Development Life Cycle
DMCA	Digital Millennium Copyright Act
IDS	Intrusion Detection System
MTTF	Mean Time To Failure
Ploc	per (source) Lines of Code

1. Introduction

This paper seek to argue that a well-defined software risk derivative market would improve the information exchange for both the software user and vendor removing the oft touted imperfect information state that is said to belie the software industry. In this way, users could have a rational means of accurately judging software risks and costs and as such the vendor could optimally apply their time between delivering features and averting risk in a manner demanded by the end user. After all, it is of little value to increase the cost per unit of software by more than an equal compensating control.

Arora, Telang and Xu [2, 3] asserted that a market based mechanism for software vulnerabilities will necessarily underperform a CERT-type mechanism. The market that they used was a game theoretic *pricing game* [14]. In the model reported, the players in the market do not report their prices[1]. These players use a model where information is simultaneously distributed to the client of the player and the vendor. The CERT model was touted as being optimal. It relies on waiting until a patch was publically released and only then releasing the patch to the public. This ignores many externalities and assumes the only control is a patch in place of other alternative compensating controls.

Consequently, the examined "market" model is in itself sub-optimal. It both creates incentives to leak information without proper safeguards and creates vulnerability black-markets. As criminal groups and selected security vendors (such as Penetration testers and IDS vendors) have an incentive to gain information secretly[2], they have an incentive to pay more for unknown vulnerabilities in a closed market. This means that a seller to one of these parties has a reputational incentive to earn more through not releasing information as the individual's reputation will be based on their ability to maintain secrecy.

This misaligned incentive creates a sub-optimal market. As a consequence, the market reported (Arora, Telang and Xu 2005; Kannan and Telang 2004) was sub-optimal. This CERT based model was an inefficient market. This does nothing to imply that all markets are less effective. The skewed incentivisation structure recommended in the paper was the source of the inefficiency and not the market itself. This simply highlights the need to allow efficient markets to develop rather than seeking to create these through design.

The other argument posed comes as a consequence of information asymmetry. It is asserted [14] that software vendors have an informational advantage over other parties. The vendor does have access to source code (which is also available for Linux and other open source providers), but it can be proved that this does not provide the levels of information asymmetry that are asserted. Software vendors have a reputational input to their value [8,20].

[1] E.g. iDefense Ltd. And other similar providers have a semi-closed market with limited information exchange.

[2] Criminal groups have an incentive to maximize the time that vulnerabilities remain unknown as this extends the time that they have to exploit these bugs. Penetration Testers etc. have similar incentives as the trade secret of an unknown zero day vulnerability can provide them with a competitive advantage. This also goes to reputational motives for both parties.

Telang & Wattal [20] did note that the market value of a software vendor is influenced through reputational costs and those vulnerabilities correlate significantly with a decrease in the companies traded price, a view supported by others [8].

> "*Vulnerability disclosure adversely and significantly affects the stock performance of a software vendor. We show that, on average, a software vendor loses around 0.63% of market value on the day of the vulnerability announcement. This translates to a dollar amount of $0.86 billion loss in market value. We also show that markets do not penalize a vendor any more if the vulnerability is discovered by a third party than by the vendor itself*."

These results demonstrate that a vendor has an incentive to minimize the vulnerabilities found in their products. If an excessive number of vulnerabilities continue to impact a vendor, their market capitalization suffers as a consequence. This justification offers strong evidence that a vendor does not have an incentive to hide information (as third party vulnerability researchers cause an equal loss in capitalization). It has to be expected that any vulnerability known by the vendor will be uncovered. If the vendor fixes this flaw before release, the cost is minimized and at the limit approaches the cost of testing, (that is a zero incremental cost to that which would be expressed later).

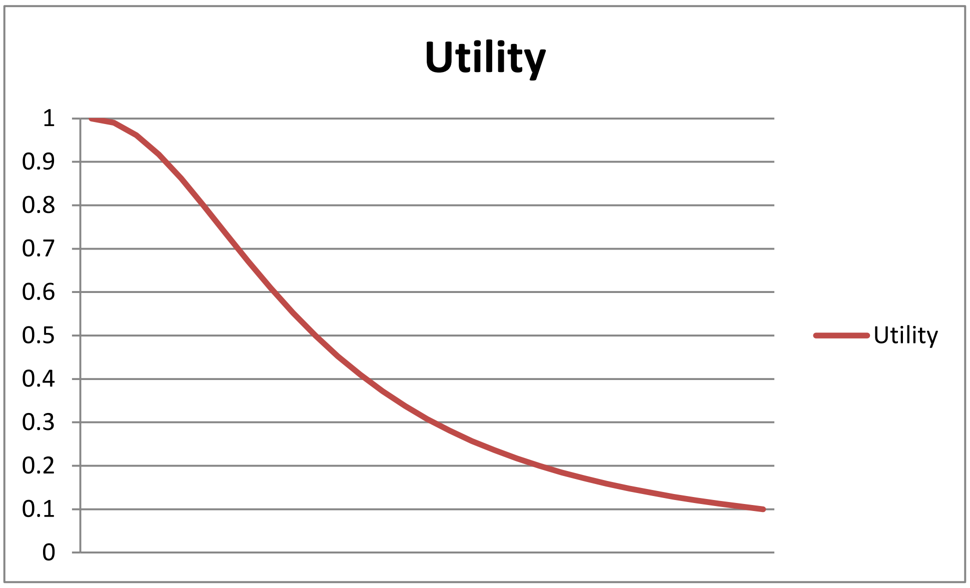

Figure 1 Decreasing utility of testing as the SDLC progresses

If the vendor discovers a vulnerability in the software they produce, the result is a *'strongly dominated'* motive to fix the bug. Hence, any remaining bugs are those that have not been uncovered by the vendor and which are less economical to find (through an increase in testing). It can thus be demonstrated that the vendor knows no more than the user at the point of software release as to the state of bugs in a product.

Testing is far less expensive earlier in the development cycle[3]. Figure 1 displays the expected utility of testing as the development progresses through an SDLC. Early in the process, the software developer has the greatest returns in testing and bug finding. As the development progresses, the returns are reduced as the process required and the costs associated with finding and correcting software vulnerabilities increases.

The utility is lowest when the software has been shipped to the user. At this point, fixing flaws is an expensive process for both the user and the vendor. This leaves the optimal solution to find as many bugs as possible as early in the development process as is feasible. This contrasts with the increasing costs of finding bugs (Figure 2). This leaves the optimal solution for the vendor based on the discovery of as many bugs as possible as early in the development process as is feasible (as a bug discovered early in the process can cost as much as 10x less [6] than one discovered later)[4]. It does not mean that all bugs or vulnerabilities will be found as the cost of finding additional vulnerabilities quickly exceeds the returns.

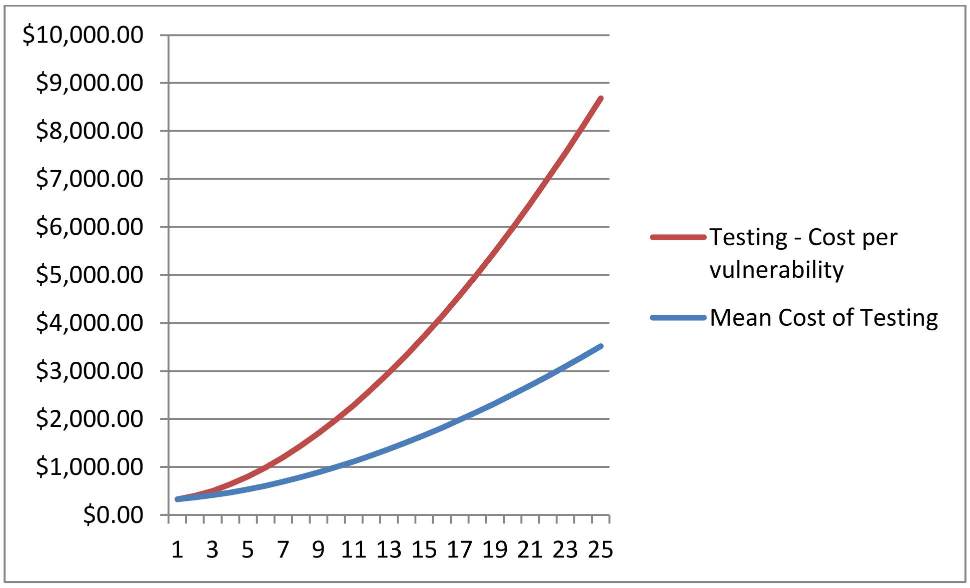

Figure 2 Each vulnerability costs more than the last to mitigate

The widespread use of Open Source software (such as Linux) and the limited differential in discovering bugs using source code reviews demonstrates that the vast majority of software flaws in commercial software are discovered early in the development process (it is highly unlikely that commercial software developers are less effective than open source developers and that the rate of errors is likely to be similar if not lower).

It has been estimated that it takes 5,000 man hours of use to discover the average software vulnerability. This amount of time creates a great amount of expense for software development when it has to be completely conducted internally. An open market on the other hand distributes this expense in an optimal manner and provides a source of information as to the true cost of the

[3] NIST http://www.cse.lehigh.edu/~gtan/bug/localCopies/nistReport.pdf, see also Impact of Software Vulnerability Announcements on the Market Value of Software Vendors – an Empirical Investigation
[4] See http://www.cse.lehigh.edu/~gtan/bug/localCopies/nistReport.pdf

vulnerability. This information is created as the true costs of developing patches and can be compared to alternatives where patching is more costly.

A vulnerability market provides information on discovery and vendor action (for instance Microsoft vs. Adobe vs. Linux etc) allowing clients to better select software vendors, mitigating the "*Market for Lemons*" [15] that has been proposed. It has been demonstrated already that software vendors do not have more knowledge of bugs than users (or these would have been fixed prior to release) and hence do not have an advantage in information asymmetry when it comes to finding software flaws that may result through product interactions.

The market for lemons requires that the vendor knows the level of flaws better than the user. To many this may seem a common sense outcome, the vendor has access to source code, wrote the program and ran the development process. This is a flawed view as we have demonstrated as it is in the vendor's interest to mitigate vulnerabilities as early as possible. More importantly, the vendor is punished for bugs [19, 22].

1.1 A legislative approach

Many information security professionals call for legislation and penalties against software vendors who have bugs in their software. This paper examines the economic impact of several approaches to enforcing software security and demonstrates that a market-based approach is the most effective.

In most existing jurisdictions, the introduction of an exculpatory rule in contracts allows for the negation of a legal ruling (the default rule) where both parties have (through the contract) expressly consented to the contrary. Just as carmaker is not liable for the life of an engine exceeding 60,000km unless there is an express warranty stating this, an express exculpatory rule for software does not exist unless the vendor offers this in a warranty clause[5].

Legislation can make many superficial exculpatory clauses either invalid or redundant. Such exclusion would include negligence. In some cases, the exculpatory clause merely restates the existing perspective of the law. Where an exculpatory clause is permissible, the final assignment of liability does not depend on the preliminary distribution of liability. If the default rule is that a software vendor is liable for software vulnerabilities, but the vendor's profitability is greater where

[5] Software vendors offer limited warranties that provide some protection for the user, but this is limited. The vendor cannot account for the actions of the user and a failure to install the software with the use of adequate controls may result in the loss.
In a decision that can be related to software flaws, *Mercedes Benz (NSW) v ANZ and National Mutual Royal Savings Bank Ltd* (unreported), the Supreme Court of New South Wales considered if a duty to avert fraud would occur in cases where there is an anticipated prospect of loss. The Mercedes Benz employee responsible for the payroll system fraudulently misappropriated nearly $1.5 million by circumventing controls in the payroll software. Mercedes Benz alleged that the defendants, ANZ and NMRB, were negligent in paying on cheques that where fraudulently procured by the employee and in following her direction. The plaintiff's claim was dismissed by the court. It was held that employers who are careless in their controls to prevent fraud using only very simple systems for the analysis of employee activities would be responsible for the losses that result because of deceitful acts committed by the organizations' employees.

it does not suffer liability, then the vendor will print a label stating that is it not liable. In the case of software, this could be a click wrap agreement.

In this paper, we investigate the economic impact of various policy decisions as a means of determining the optimal distribution of costs and liability when applied to information security and in particular when assigning costs in software engineering.

1.2 Risk assignment and Software Contracts

In economic terms, we want to assign liability such that the optimal damage mitigation strategy occurs. The victim will mitigate their damages where no damages for breach apply in respect of the optimal strategy and payoffs. The rule that creates the best incentives for both parties is the doctrine of avoidable consequences (marginal costs liability).

Mitigation of damages is concerned with both the post-breach behaviors of the victim and the actions of the party to minimize the impact of a breach. In a software parlays', this would incur costs to the user of the software in order to adequately secure their systems. This again is a trade-off. Before the breach (through software failures and vulnerabilities that can lead to a violation of a system's security), the user has an obligation to install and maintain the system in a secure state. The user is likely to have the software products of several vendors installed on a single system. Because of this, the interactions of the software selected and installed by the user span the range of multiple sources and no single software vendor can account for all possible combinations and interactions.

Any pre-breach behavior of the vendor and user of software needs to incorporate the capability of the vendors to both minimize the liability attached to their own products, as well as the interactions of other products installed on a system. It is feasible to deploy one of several options that can aid in the minimization of the effects of a breach due to a software problem prior to the discovery of software vulnerabilities, these include:

1. The software vendor can implement protective controls (such as firewalls)
2. The user can install protective controls
3. the vendor can provide accounting and tracking functions

The following steps further facilitate in minimizing the effects of software vulnerabilities:

1. The vendor can employ more people to test software for vulnerabilities
2. The software vendor can add additional controls

Where more time is expended on the provision of software security by the vendor (hiring more testers, more time writing code etc), the cost of the software needs to reflect this additional effort, hence the cost to the consumer increases. This cost is divisible in the case of a widely deployed Operating System (such as Microsoft Windows) where it is easy to distribute the incremental costs across additional users. Smaller vendors (such as small tailored vendors for the Hotel accounting market) do not have this distributional margin and the additional controls could result in a substantial increase in the cost of the program.

This is not to say that no liability does or should apply to the software vendor. The vendor in particular faces a reputational cost if they fail to maintain a satisfactory level of controls or do not respond to security vulnerabilities quickly enough or suffer too many problems. The accumulation of a large number of software vulnerabilities by a vendor has both a reputational cost to the vendor as well as a direct cost to the user (these costs include the time to install and the associated downtime and lost productivity). Consequently, a user can conduct an investigation into the accumulation of software vulnerabilities and the associated difficulty of patching or otherwise mitigating flaws prior to a purchase> These costs are thereby assigned to new vendors even if they experience an exceptionally low rate of patching/vulnerabilities. The user can act with knowledge of costs where it is efficient for them to do so. As users are rational in their purchasing actions, they will incorporate the costs of patching their systems into the purchase price .

The probability of a vulnerability occurring in a software product will never approach zero. Consequently, it follows that the testing process used by the vendor is expressible as a hazard model [5]. In this, it is optimal for the vendor to maximize their returns such that they balance the costs of software testing against their reputation [22].

The provision of a market for vulnerabilities leads to a means of discovering the cost of finding vulnerabilities as an optimal function. In this way, the software vendor maximizes their testing through a market process. This will result in the vendor extending their own testing to the point where they cannot efficiently discover more bugs. The prices of the bugs that sell on market are definite and the vendor has to pay to either purchase these from the vulnerability researcher (who has a specialization in uncovering bugs) or increase their own testing. The vendor will continue to increase the amount of testing that they conduct until the cost of their testing exceeds the cost of purchasing the vulnerability. The potential pool of vulnerability researchers is larger than the potential pool of in-house testing roles. The pool of vulnerability researchers would also increase unto a point where the cost of training to become a vulnerability researcher matches the demand of the consumer to fund vulnerabilities.

This market also acts as an efficient transaction process for the assignment of negligence costs. The user still has to maintain the optimal level of controls that are under their influence (installation, patching frequency etc), whilst the vendor is persuaded to pay the optimal level of costs for testing and mitigation.

2. Software Derivative Markets

One possible solution to the limited and sub-optimal markets that currently exist would be the creation of Hedge funds for software security. Sales in software security based derivatives could be created on forward contracts. One such solution is the issuing of paired contracts (such as exist in short sales of stocks[6]). The first contract would be taken by a user and would pay a fixed amount if the software has suffered from any unmitigated vulnerabilities on the (forward) date specified in the contract. The paired contract would cover the vendor. If the vendor creates software without flaws (or at least mitigates all easily determinable flaws prior to the inception of the contract) the contract pays them the same amount as the first contract.

[6] Short selling involves an investor anticipating a decrease in the price or an item. This involves selling a chattel that the investor does not own through a contract agreement. If the goods sell higher than anticipated, the investor losses money as they have to purchase the goods at the (now higher) market rate.

This is in effect a 'bet' that the software will perform effectively. If a bug is discovered, the user is paid a predetermined amount. This amount can be determined by the user to cover the expected costs of patching and any consequential damages (if so desired). This allows the user to select their own risk position by purchasing more or less risk as suits both the risk tolerance and the nature of the user's systems.

Such a derivative (if an open market is allowed to exist) would indicate the consensus opinion as to the security of the software and the reputation of the vendor. Such an instrument would allow software vendors and users to hedge the risks faced by undiscovered software vulnerabilities. These instruments would also be in the interest of the software vendor's investors as the ability to manage risk in advance would allow for forward financial planning and limit the negative impact that vulnerability discovery has on the quoted prices of a vendors capital.

2.1 Selective Survival

If the vendor creates the low feature version for $200 and the full featured secure version for $800, the user can choose the level of protection. Taking a survival model of Windows 98 and one for Windows XP [7], the client can calculate the expected difference from each product. If Windows 98 has an expected compromise rate (without other compensating controls) of once per 6.2 days and Windows XP has an expected rate of 52.5 days when the firewall is enabled[7] we can calculate the cost per incident. If the user has a mean cost per incident of $320[8], we see that the Windows 98 option has an expected annual cost to the organization of $19,040 ($18,840 damage plus $200) whereas Windows XP has an expected cost of $3,825 ($3,025 damage plus $800).

Hence, most companies with a need to secure systems selected the more secure option and installed Windows XP over Windows 98. The initial costs did not reflect the overall costs to the organization. Likewise, home users (who rarely calculate the costs of compromise) where more likely to install the cheaper option.

To extend the analogy, assume that a software vendor can create a perfect version of software for a mean cost of $20,000[9] and the flawed version at the standard costs of under $1,000. Where the expected damages do not exceed $19,000 per copy of software, we see that it remains in the user's interests to select the lower security option as the expected losses are lower. More generally, for an organization with ***n*** users and an expected cost $C_S = nP_S$ (where P_S is the cost per user of the secure software product) against $C_I = nP_I$ (with P_I being the cost per user of the insecure software product), the cost of deploying either option is based on the expected losses for each option with a loss function defined as $D_S < D_I$. As such, if $\left(C_S - D_S\right) < \left(C_I - D_I\right)$ it is in the

[7] In each case we are presuming that the Centre for Internet Security (www.cisecurity.org) standards for installing the software have been reasonably met.
[8] The quantitative data that this statement is derived from is to be released in a follow-up paper next year.
[9] Which is an extremely low estimate.

interest of the company to take the secure option. Where $\left(C_S - D_S\right) > \left(C_I - D_I\right)$, the economical solution is to install the less secure version of the software and self-insure against the loss.

As a means of maximizing the allocations of risk and costs, the vendor could offer liability-free and full-liability versions of their product, the later sold at a reduced price. The vendor could then control their risk using a hedging instrument[10]. A market evaluation of the risk being traded would provide better information than that which the vendor has alone[11]. With this in mind, the user could also purchase the same derivative as the vendor. The cost to the user to purchase the software plus risk insurance would be less than purchasing the "more secure" version from the vendor as the vendor would hedge the risk it faces and add a transactional cost as well as a return (at the firms IRR).

Both parties are better off where the vendor produces the optimized version of their product and where the vendor does not assume liability for bugs.

Simply put, the vendor will provide a warranty (which acts as an insurance policy for the user) in cases where it can charge more for the provision of the warranty than it costs to provide the warranty. This cost could be either from a hedging instrument or through an investment in more testing. However, it must be noted that no increasing testing will make software invulnerable to all attacks and remove the ability for flaws to be discovered. As such, the cost of the hedged derivative will decrease, but will not go to zero.

The argument against this form of market is imperfect information. Many people argue that the software vendor has more knowledge than the user. This is untrue for several reasons. The vendor has no incentive to hide vulnerabilities as each vulnerability impacts the share price of the vendor through a decrease in capital [20]. Next, the vendor would be competing on the same derivative markets as the users. The vendor’s knowledge would be quickly and efficiently transferred through the market mechanism. The end result is that a derivatives based strategy does not allow for information asymmetry and would disprove entirely the assertion that software vendors operate a "*market for lemons*".

From this we can see that the lack of liability does not require that the software vendor does not have the appropriate incentive to provide the optimal amount of testing and hence security (as well as to add the optimal levels of external controls) to the user. If more testing is cost effective in economically providing more secure software, that is if the additional testing is cost effective, the software vendor will conduct this testing whether it is liable for security flaws or not.

The incentive to provide security is no different than the incentive to provide other characteristics and features. A vendor will add software extensions that can reduce the overall security of a system if the user values these to a greater extent than their costs.

---

[10] As discussed earlier

[11] Not only does the market pooled knowledge feed into the price of the derivative, but the vendor's knowledge of their own product does as well.

Safety (and security in general) is a tie-in good. Amenities cost something. Vendors provide amenities if consumers (users) are willing to pay the cost of obtaining these.

"*Cars come with batteries and tires, but not always with stereo equipment or antitheft devices and rarely with driving gloves and sunglasses*" [23].

What is demonstrated by those organizations with a defensive (and hence secure) coding regime is a lower variability in output. The mean coding time will remain similar, but the test time can be expected to be distributed differently for the organization that codes with a defensive posture than that of the organization that leaves too little time for testing. This increase in the standard deviation of produced results (or variability) increases the risk to the all parties (vendor and user).

2.2 Imperfect Information

At present, it can be demonstrated that neither side (the user or the vendor) is informed as to the true risks and costs of software. The vendor does not know the situation of the user (for the most part) and the user may not have an easily accessibly means of determining the risk from software. Worse, neither side is likely to know the risk posed from integrating multiple software products.

This asymmetry can be modeled. If we take two hypothetical firms, SecureSoft and BuggySoft each with 50% of the market for their software products and give SecureSoft a mean time between failures[12] (MTBF) of 86 days and BuggySoft a MTBF of 17 days. In each case, the cost of a compromise is determined to have a mean of $400 (with).

Next, assume that the average user (and purchaser of the software) cannot tell the difference. This is that the user sees a MTBF of 52 days for either software vendor. This situation is known as 'adverse selection' in economics. If the users are liable and they self-insure, they will not be able to determine the relative level of vulnerabilities with respect to SecureSoft or BuggySoft. The cost of purchase will only be probabilistically determined (although, there are only small odds that an overlap event[13] will occur) when the software vulnerability is uncovered.

If the software vendor were liable for more than one (1) vulnerability each 90 days, we would expect SecureSoft to breach around 40% of the time for an expected cost of $1,622 per annum. BuggySoft on the other hand would be expected to breach 100% of the time for an expected cost of $8,588 per annum. The difference in value of $6,966 is what BuggySoft would have to pay to its customers each year.

As such, SecureSoft could market itself as a more secure option. If the user-base did not believe that BuggySoft and SecureSoft are different, SecureSoft could offer $6,965 worth of liability insurance to its users whereas BuggySoft would not be able to match this price (all other aspect being equal). As a result, even an argument for enforced liability insurance has a weakness. In the absence of liability legislation, the more secure software vendor would still desire to offer

[12] For simplicity, we will assume that this is a vulnerability leading to a catastrophic flaw such as a remote "root" compromise. It is possible to model non-destructive failure processes using “sickness – recovery – death” survival models, but these are outside the scope of this paper.
[13] Where the survival time of the first company falls in the expected range of the other company.

insurance to its users where it is cost effective to do so. By doing this, SecureSoft could add benefit to its users by insuring for loss to a level that is not cost effective for BuggySoft. This price differential would gradually win business for SecureSoft and increase its market share.

The consequence is that although users see an equal distribution of problems initially, over time, the information asymmetry decreases and the market starts to favor the most secure provider. The difficulty in these circumstances is not comparing which product is more secure, but is an issue of trading security against features. If BuggySoft added a management interface included with its software package that users valued at $8,000, they would be able to outsell SecureSoft even with the insecurities and added risk.

From this example, we see the inherent issue with software security. Users value many aspects of a product and value these against the overall value that they assign to the product. As a result, an insecure product with saleable features will commonly sell for more than the secure version (even where users do not actually use the added features).

2.3 Optimal Derivatives Design under Dynamic Risk Measures

The game theoretic approach to this can be modeled looking at the incentives of the business and programming functions in the organization. Programmers are optimists [6]. As Brooks [6] noted, "*the first assumption that underlies the scheduling of systems programming is that all will go well*". Testing is rarely considered by the normal programmer as this would imply failure. However, the human inability to create perfection leads to the introductions of flaws at each stage of development.

In the model presented by Brooks [6], as a program moves to a "Programming Product" and then to "a Programming Systems Product"[14], there are incremental additions that extend the estimates of the programming team. At each phase these can be expressed as a function of effort expressed in the lines of code, l_c. We can express the mean effort $\overline{x}_{l_c}$ required to code a number of lines and the variability[15], σ_{l_c} in achieving this outcome. This allows us to represent the coding effort as a representation of the stages of development:

$$F(x) = 9H\left(\overline{x}_{l_c}, \sigma_{l_c}\right)G\left(\overline{x}_{l_c}, \sigma_{l_c}\right) + \varepsilon \qquad (1)$$

In (1), $H\left(\overline{x}_{l_c}, \sigma_{l_c}\right)$ is the function of systematizing [6, p6] the code. The expression $G\left(\overline{x}_{l_c}, \sigma_{l_c}\right)$ is the productization [6, p6] of the code.

The derivative would require the firm to publish their productivity figures;

[14] Through the addition of a "Programming System" with the addition of Interfaces and a System Integration Phase.

[15] Standard Deviation (where $\sigma = Variance^2$)

1. Lines of code per programmer (and standard Deviation); l_c, $\overline{x}_{l_c}$ and σ_{l_c}
2. Bug-incidence figures; $\overline{b}_{l_c}$ and σ_b
3. Estimating and project rules/methodology
4. Software design methodology
5. Secure Programmer measures[16]. $\overline{t}$ and σ_t

Many see this as proprietary data and would be loathe to share, but as more in the market take-up the use of such a derivative in managing their own risk, the incentives for others to follow increase. We can also demonstrate that as $\overline{t} \to \max_t$ and $\sigma_t \to 0$ that the volatility in coding σ_{l_c} and σ_b decrease with the number of reported bug incidents $\overline{b}_{l_c} \to 0$ as $\overline{t} \to \max_t$. More metrics would be required based on the methodologies and coding practices used with different computer languages expected to exhibit different responses[17].

As the skill of the programming team ($\overline{t} \to \max_t$ & $\sigma_t \to 0$) increases through group training, secure coding practices and in-line testing, the volatility in coding $\sigma_{l_c} \to 0.$ As the incremental returns on $\overline{t}$ diminish as $\overline{t} \to \max_t$ and the cost of training continue to grow linearly[18], we can see that the optimal position for any software vendor is to have a skilled team that maximizes returns. At this point, additional training becomes economically unviable and does not create further returns for the organization[19]. We also see that it is in the interest of the firm to minimize the variance in skill levels between programming staff (aim to have $\sigma_t \to 0$). This of course adds costs as junior staff would be less productive in that they would have to work with more senior staff[20] in order to both cross train and to ensure that the junior employees maintain an acceptable level of production and testing.

[16] e.g. SANS Coding Methodology.
[17] Which would be quantified through the free market method.
[18] That is, $Cost(t) = at + b$ where there is an initial cost plus a linear increase in the costs of developing the programmer skills.
[19] This investment may still be made by the individual programmer with returns to the organization and also for further personal gains (investment in personal capital).
[20] Also reducing the output of the senior staff.

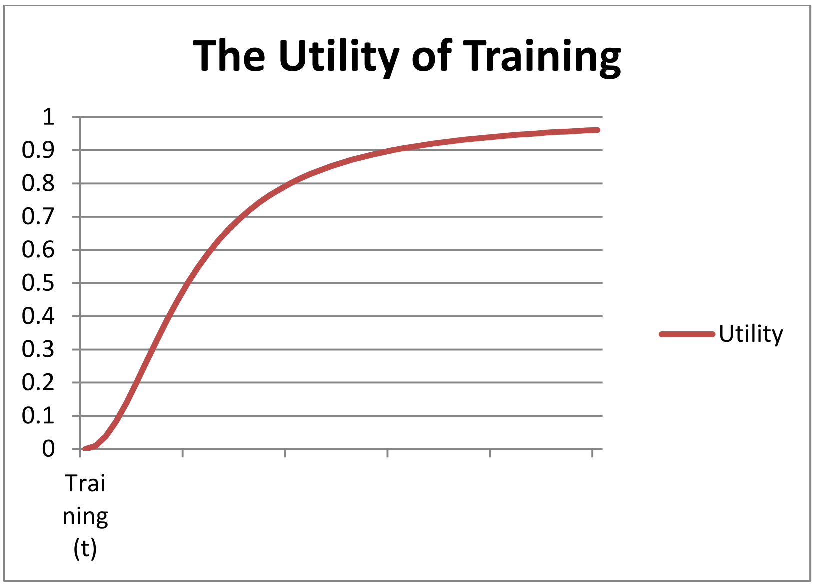

Figure 3 **Training Software developers in Security adds utility**

This exercise of joining junior and senior staff would create a return function depicted by Brookes [6, p19] for a "*Time versus number of workers - task with complex interrelationships*". The output of the junior/senior staff unit would be lower than the senior staff level (initially) but would lower variability and increase the skill levels of the junior staff and hence over time $\sigma_t \to 0$. Due to staff turn-over[21] and the increasing costs of maintaining more skilled programmers, this would also have a negative feedback on the vendor's production costs. This would also need to be optimized rather than maximized if the vendor is to maximize returns.

2.4 No More Lemons

The fallacy of the mainstream analysis of deflation can be used to demonstrate the fallacy of aligning a market for lemons to computer software. This game model has people refrain from spending when they anticipate falling prices. As people expect prices to fall, they believe that they will get a better price if they consume later (or that they will get more secure software for the same price).This drop in consumption results in a consequential price fall, and the whole cycle repeats. The argument made is that prices will spiral uncontrollably downward.

Robert Murphy [14] addressed this issue and demonstrated its fallacy:

> *One could construct an analogous argument for the computer industry, in which the government passes regulation to slow down improvements in operating systems and processing speed. After all, how can computer manufacturers possibly remain viable if consumers are always waiting for a faster model to become available? … The solution to this paradox, of course, is that consumers do decide to bite the bullet and buy a computer, knowing full well that they would*

[21] F. J Corbato' (MIT) – "*A long duration project needs to incorporate a turnover rate of at least 20% per annum. New hires need to be technically trained and also require time to be able to effectively integrate into the existing team*".

be able to buy the same performance for less money, if they were willing to wait… (There's no point in holding out for lower prices but never actually buying!) (pp. 68–9)

Some users do wait for patches to be proven and others use "*bleeding edge software*" (and some never patch). The model is more rightly based on a distribution of users centered on an install time slightly after the release date.

This model can help us combat George Akerlof's lemons model[22] of the used-car market has been applied to the computer industry. Akerlof argued that asymmetric information would result in the owners of good used cars being driven from the market. Contrary to the equilibrium suggested in this model, good used cars sell. One can state that just because owners of good cars may not be able to obtain as high a price as they would like, it does not follow that they will refuse to sell at all.

Likewise, just as users do not know the state of security or how many bugs exist in software, software is still sold.

The "real balance effect": as prices fall, the value of money rises.

People's demand to hold money can be satisfied with less money as price deflation occurs. This in part explains why people will eventually spend, even when they expect prices to continue to fall. The same process applies to software. Users see value in the features and processes they purchase software for. These are offset against the costs which include both the monetary costs as well as the costs of maintenance (patching) and security (and other) flaws. Failure has a cost and this is incorporated in to the price of software to the user. The result is that users buy and install software even when they expect more bugs/vulnerabilities to be found.

3. Conclusion

Just as car dealers buff the exterior and detail the upholstery of a used car, neglecting the work that should be done on the engine, software vendors add features. Most users are unlikely to use even a small fraction of these features, yet they buy the product that offers more features over the more secure product with fewer features. The issue here is that users buy the features over security. This is a less expensive option for the vendor to implement and provide.

The creation of a security and risk derivative should change this. The user would have an upfront estimate of the costs and this could be forced back to the software vendor. Where the derivative costs more than testing, the vendor would conduct more in-depth testing and reduce the levels of bugs. This would most likely lead to product differentiation (as occurred in the past with Windows 95/Windows NT). Those businesses will to pay for security could receive it. Those wanting features would get what they asked for.

[22] See 15 for various derived models.

It is argued that software developers characteristically do not correct all the security vulnerabilities and that known ones remain in the product after release. Whether this is due to a lack of resources or other reasons, this is unlikely to be the norm and would be rectified by the market. The cost of vendors in share price [1] and reputational losses exceed the perceived gains from technical reasons where the fix might break existing applications. The application is already broken in the instance of a security vulnerability.

Users could still run older versions of software and have few if any bugs. The issue is that they would also gain no new features. It is clear that users want features. They could also choose to use only secure software, but the costs of doing so far outweigh the benefits and do not provide a guarantee against the security of a system being compromised. As such, the enforced legislation of security standards against software vendors is detrimental. A better approach would be to allow an open market [4] based system where vendors can operate in reputational and derivative markets.

At the end of any analysis, security is a risk function and what is most important is not the creation of perfectly security systems, but the correct allocation of scarce resources. Systems need to be created that allow the end user to determine their own acceptable level of risk based on good information.

4. References

1. Arora, A. & Telang, R. (2005), "Economics of Software Vulnerability Disclosure", IEEE Security and Privacy, 3 (1), 20-2
2. Arora, A., Telang, R. & Xu, H. (2004) "Optimal Time Disclosure of Software Vulnerabilities", Conference on Information Systems and Technology, Denver CO, October 23-2
3. Arora, A., Telang, R. & Xu, H. (2008), "Optimal Policy for Software Vulnerability Disclosure", Management Science 54(4), 642-6
4. Bacon, D. F., Chen, Y., Parkes, D., & Rao, M. (2009). A market-based approach to software evolution. Paper presented at the Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.
5. Beach, J. R., & Bonewell, M. L. (1993). Setting-up a successful software vendor evaluation/qualification process for `off-the-shelve' commercial software used in medical devices. Paper presented at the Computer-Based Medical Systems, 1993. Proceedings of Sixth Annual IEEE Symposium on.
6. Brookes, F. (1995) "The Mythical Man-Month". Addison-Wesley
7. Campodonico, S. (1994). A Bayesian Analysis of the Logarithmic-Poisson Execution Time Model Based on Expert Opinion and Failure Data. IEEE Transactions on Software Engineering, 20, 677-683.
8. Cavusoglu, H., Cavusoglu, H. & Zhang, J. (2006) Economics of Security Patch Management, The Fifth Workshop on the Economics of Information Security (WEIS 2006)
9. Cohen,. J. (2006). Best Kept Secrets of Peer Code Review (Modern Approach. Practical Advice.). Smartbearsoftware.com
10. de Villiers, M (2005) "Free Radicals in Cyberspace, Complex Issues in Information Warefare" 4 Nw. J. Tech. & Intell. Prop. 13, http://www.law.northwestern.edu/journals/njtip/v4/n1/2
11. Dijkstra, E. W. (1972). "Chapter I: Notes on structured programming Structured programming" (pp. 1-82): Academic Press Ltd.
12. Kannan K & R Telang (2004) 'Market for Software Vulnerabilities? Think Again.' Management Science.

13. Mills, H. D. (1971) "Top-down programming in large systems", Debugging techniques in large systems, R. Rustin Ed., Englewoods Cliffs, N.J. Prentice-Hall
14. Murphy, R. & Regnery, P. (2009) “The Politically Incorrect Guide to the Great Depression and the New Deal”.
15. Nissan, N., Roughgarden, T., Tardos, E. & Vazirani, V. (Eds.) (2007) “Algorithmic Game Theory” Cambridge University Press, {P14, Pricing Game; P24, Algorithm for a simple market; P639 Information Asymmetry).
16. Nizovtsev, D., & Thursby, M. (2005) “Economic analysis of incentives to disclose software vulnerabilities”. In Fourth Workshop on the Economics of Information Security.
17. Ounce Labs, 2. http://www.ouncelabs.com/about/news/337-the_cost_of_fixing_an_application_vulnerability
18. Ozment, A. (2004). “Bug auctions: Vulnerability markets reconsidered”. In Third Workshop on the Economics of Information Security
19. Perrow, C. (1984/1999). Normal Accidents: Living with High-Risk Technologies, Princeton University Press.
20. Telang, R., & Wattal, S. (2004) “Impact of Software Vulnerability Announcements on the Market Value of Software Vendors – an Empirical Investigation” http://infosecon.net/workshop/pdf/telang_wattal.pdf
21. Turing, A (1936), “*On computable numbers, with an application to the Entscheidungsproblem*”, Proceedings of the London Mathematical Society, Series 2, 42 pp 230–265
22. Weigelt, K. & Camerer, C. (1988) “Reputation and Corporate Strategy: A Review of Recent Theory and Applications” Strategic Management Journal, Vol. 9, No. 5 (Sep. - Oct., 1988), pp. 443-454, John Wiley & Sons
23. Donald, D. (2006), "Economic Foundations of Law and Organization" Cambridge University Press