Journal of Physical Sciences and Innovation *Volume 2, September 2010*

www.cenresin.org

ECONOMIC MODELS FOR SOFTWARE SECURITY

Akwuwuma Veronica and Egwali Annie
Department of Computer Science
University of Benin, Benin City, Nigeria
E-mail: vakwukwuma@yahoo.com; egwali.annie@yahoo.com

ABSTRACT

The economics of software security is an evaluation of the cost and benefits of adding security to software. Most firms are mainly concerned with making software functional without paying much consideration to security because of the rigors of adding security to software and because buyers have no low cost method of ascertaining quality. These nonchalant practices will only give attackers an upper hand in the race for compromising system software. It is crucial therefore that software developer protect their customers by embedding security and confidentiality into their software. Security should be a factor in software development undermining the cost. In this paper, we therefore focus on the economics of building security properties into software application. To cover the entire dimension of the economics of software security, we incorporate cost-benefit analysis models[1] to incur the cost of adding security properties into software development. We also expanded our security framework to integrate the security properties[2] and incorporated encryption, TCP/IP hardening and Buffer overflow checks.

Keyword: Cost, Security, Software, Economics, Attacks

INTRODUCTION

In our world today, the presence of door locks, gated communities, guard dogs, access cards and identification badges are all testaments to a physically insecure world. Likewise, the need for similar protective mechanisms is no less significant as our digital world is an every bit as insecure as our physical one. The public has become more conscious of the need for software security and so too has the government. In a digitally secured world, there would be no need for investing in secure software but the reverse is the case. These days, most software developers are concerned more about functionality alone and disregarding security because many firms are not willing to invest in secure software development and so the major software deployed in our digital world is insecure. This is coupled with the fact that buyers has no low cost method for ascertaining quality and the cost of insecure code is often borne by someone else (a negative externality). The importance of building in security at an early level in the software development life cycle was highlighted[3], stating that security should be implemented at the requirements level of the software development life cycle (SDLC) and after deployment, and that knowledge gained by understanding attacks and exploits should be cycled back into the SDLC.

The central principle of the strategic software design thought us that software design is an investment activity, involving the expenditure of valuable resources, whose goal is to maximize value added net to costs on a scale appropriate to a given context. Thus the value of a software system reflects both its fitness to meet current needs and its capacity to evolve to sustain fitness as conditions change over time. Thus the security level provided by developers is consistently lower than it should be. For one to invest in secure software building, threats to software security must be understood.

Presently there are several threats to software systems [4]. Among which are viruses, worms and Trojans. It was posited that viruses are programs that threaten the security of most software product[5]. He defined a virus as a self-replicating program that injects itself to a host and then keeps attaching itself to other programs in a system it infects thereby causing damage. As for worms, it copies itself from computer to computer causing damages. Trojans are also programs that pretend to be useful but that either contain harmful code or are just plain harmful[6].

Despite these varieties of software threats, many software developed lack security features to battle with the ever attacking malicious software. General security principles that should be applied in secure software development were outlined as[2]:

- Authentication *(who a user is)*
- Authorization *(what a user can do)*
- Confidentiality (what a user can see)
- Non-repudiation *(did a user really perform an action)*
- Availability *(the system is ready for user activity)*

It was further stressed that because one would not find all the bugs in the software, or the entire inherent vulnerabilities and will have errors of omission and oversight, therefore every secure designer should define threats, assess the impact of vulnerably in the software and then implement a countermeasure[2]. Unfortunately, these security principles pose a challenge during the actual design of software. The more secure a system is, the less usable it usually becomes and that is normal and cannot be counteracted. For software systems to stay usable and attractive to customers, tradeoffs have to be made in terms of cost, usability, security, portability and memorability. Also, the testing process is time consuming. The developer needs to probe the system like an attacker would and have to make use of white box testing, which can increase the cost of production considerably as believed by many software developers. As for security breaches, current practices grossly underestimate the cost of security breaches, which leads to under investment in security.

This paper therefore focuses on the economics of building security properties into software application. To cover the entire dimension of the economics of software security, we incorporate the cost-benefit analysis models[1] to incur the cost of adding security properties into software development. We also expanded our security framework to integrate the security properties[2] and incorporated encryption; TCP/IP hardening and Buffer overflow checks. Section 2 contains a review of related materials. The cost-benefit analysis model proposed was discussed in section 3; this is followed by our conclusions.

Related Works

There are several cost analysis models which have been used in making decisions regarding secure or insecure software products. A cost-benefit method of selecting different software development processes that essentially tried to trade off functionality and delay, was proposed[7]. It was asserted that it is difficult to compare and contrast models of software development because their proponents often use different terminology, and the models often have little in common except their beginnings (marked by a recognition that a problem exists)

and ends (marked by the existence of a software solution)[7]. A framework provided was limited to serve just as a basis for analyzing the similarities and differences among alternate life-cycle models; as a tool for software engineering researchers to help describe the probable impacts of a life-cycle mode; and as a means to help software practitioners decide on an appropriate life-cycle model to utilize on a particular project or in a particular application area.

An optimization model was proposed in which the *total benefits (B)* and *total costs (T_{cost})* associated with different levels of information security (S) activities are accessed on an ex ante basis, the goal was to implement security procedures up to the point where the difference (*gain (G)*) were maximum[8]. For expatiation, let us assume that the dollar value of the benefits from information security activities increase at a decreasing rate. Also on the cost side, let us assume the variable portion of such costs are initially increasing at a decreasing rate, but eventually increase at an increasing rate, relative to increasing levels of information security. Furthermore, we also assume that once a decision (either explicit or implicit) is made to have some level of information security, a firm will incur a lump-sum fixed cost (for such things as personnel and software items). The above idea can be illustrated in terms of a graph shown in figure 1. As shown in the figure 1, S* is the point where the difference between the benefits and costs is greatest. Thus, the ex ante goal would be to implement information security procedures up to that point. In more formal terms, the value S* that maximizes G(S) = B(S) – T_{cost} (S) is characterized by the condition given in equation 1 shown below:

$$\frac{dG}{dS} = \frac{dB}{dS} - \frac{d\,T_{cost}}{dS} = 0 \qquad (1)$$

i.e.

$$i.e. \quad \frac{dB}{dS} = \frac{d\,T_{cost}}{dS} \quad \text{or marginal benefits = marginal costs}$$

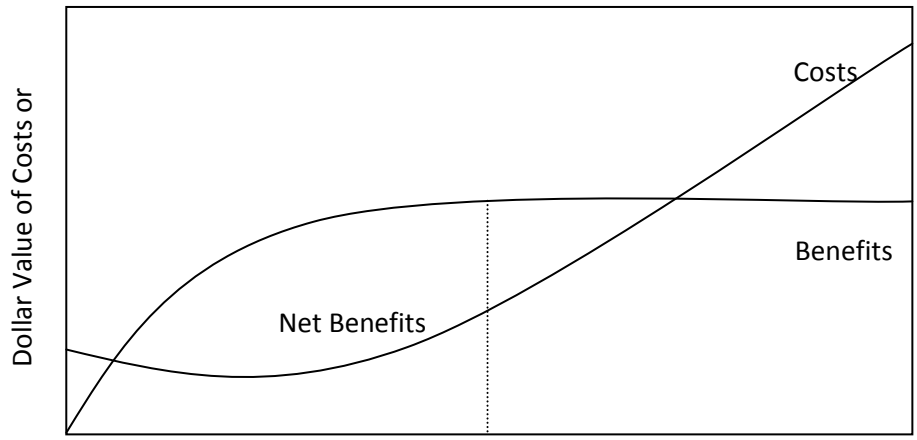

Figure 1: Level of Information Security (Gordon, 2003)

An alternative way to view the above discussion is to consider the benefits of information security as costs of not having such security. Under this view, the goal would be to minimize the sum of the costs associated with information security breaches and the costs associated with implementing information security procedures. Figure 4 provides a graph illustrating this approach. Here again, S* would represent the optimal level of information security to an organization.

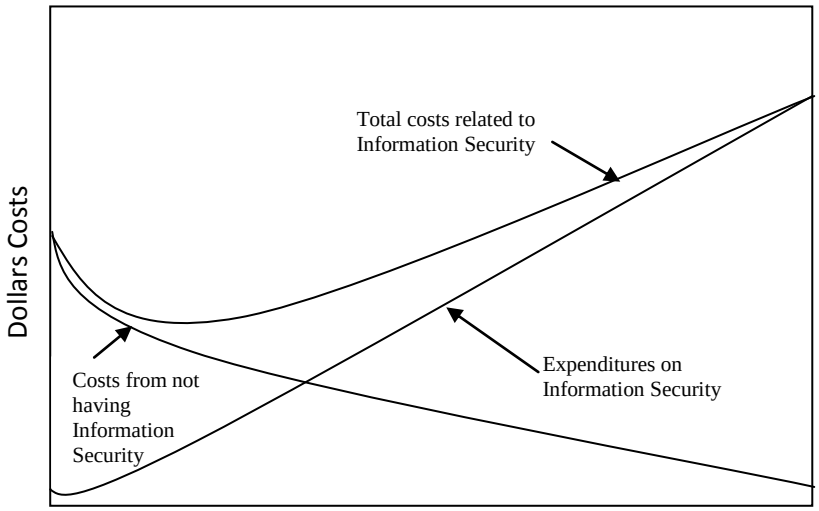

Figure 2: Level of Information Security (Gordon, 2001)

A framework was proposed that estimates the cost of software security breaches as the loss in market value of the firm due to security breach announcement[9]. Security breaches signal to the market a lack of concern for customer privacy and/or poor security practices within the firm. These signals in turn lead investors to question the long-term performance of the firm. In efficient markets, investors believed to revise their expectations based on new information in announcements and reflect those expectations in the market value of the firm. Using investors' reactions in capital markets as a proxy to estimate security breach cost, findings revealed that the publicly traded breached firms, on average, lost approximately 2.1% of their market value within two Days surrounding the security breaches. This percentage translated into a $1.65 billion average loss in market capitalization per breach based on the mean market value of firms in their data set. The average market value loss increased over time, which suggests that investors are becoming more aware of the security issues and are likely to penalize firms more for security breaches. A shortfall in the framework is that the magnitude of the loss was the same across different breach types.

[10] presented a model of network security and software piracy and contrasted two policies that a software vendor could enforce: (1) restriction of security patches only to legitimate users or (2) provision of access to security patches to all users whether their copies are licensed or not. They found that when the software security risk is high and the piracy enforcement level is low, or when tendency for piracy in the customer population is high, it is optimal for the vendor to restrict unlicensed users from applying software security patch restrictions is optimal for the vendor only when the piracy enforcement level is high.

[11] presented a net present value (NPV) model in which the present value is the value of a stream of future cash flows, negative or positive. The value of each cash flow needs to be adjusted for risk and the time value of money. The net present value (NPV) includes all cash flows including initial cash flows such as the cost of purchasing an asset, whereas a present value does not. The simple present value is useful where the negative cash flow is an initial one-off, as when buying a security. A discount rate needs to be used to adjust for risk and time value, and it is applied like this:

$NPV = CF0 + CF1/(1+r) + CF2/(1+r)2 + CF3/(1+r)3 \ldots$

Where CF1 is the cash flow the investor receives in the first year, Cf2 the cash flow the investor receives in the second year etc. The series will usually end in a terminal value, which is a rough estimate of the value at that point. It is usual for this to be sufficient far in the future to have only a minor effect on the NPV, so a rough estimate, usually based on a valuation ratio, is acceptable. Periods other than a year could be used, but the discount rate needs to be adjusted. Assuming an annual discount rate is used initially, then to adjust to another period, given a rate i, annual rate r, for a period x, where x is a fraction (e.g., six months = 0.5) or a multiple of the number of years, then the following expression suffices:

$$i + 1 = (r + 1)x$$

To use discount rates that vary over time (where r1 is the rate in the period, r2 the rate in the second period etc.) we would have to resort to a more basic form of the calculation:

$$NPV = CF0 + CF1/(1+r1) + CF2/((1+r1) \times (1+r2)) + CF3/((1+r1) \times (1 + r2) \times (1+ r3)) \ldots$$

This is tedious to calculate by hand but is fairly easy to implement in a spreadsheet, after making such calculations on a secure or insecure software sample, a decision can be made as to whether a software product should be accepted or rejected based on the results.

PROPOSED COST-BENEFIT ANALYSIS MODEL

Cost-benefit analysis is a term that refers both to helping to appraise, or assess the case for a project or proposal, which itself is a process known as project appraisal; and an informal approach to making decisions of any kind. Under both definitions the process involves, (explicitly or implicitly), weighing the total expected costs against the total expected benefits of one or more actions in order to choose the best or more profitable option in software production. The formal process is often referred to as either CBA (Cost-Benefit Analysis) or BCA (Benefit-Cost Analysis). The decisions as to whether to practice secure software development or not could be gotten from the results of the Cost-Benefit Analysis.

Some formulas were proposed that can be used for calculating the benefits and costs of adding security properties to software[1]. To calculate the benefits gained from adding security to the software project, the following information was gathered:

(i) *Program Size:* The program's size in number of source lines of codes.
(ii) *Bug Frequency*: The number of bugs both security and non-security bugs) that appear in the program per thousand source lines of code.
(iii) *Cost Incurred from Bugs*: The overall cost per bug.

The *program size* is a fairly easy number to ascertain from anyone on the development team, and recent research indicates that it may be possible to roughly determine the number of lines of code from a compiled program based on file size. *Bug frequency* is a combination of the total number of non-security bugs plus the total number of security bugs that occur per thousand line of code. Research currently indicates that the number of security bugs per thousand lines of code ranges from around 1 to 6. *The costs incurred from bugs* are divided into pre-release costs and post-release costs. The following information is required to determine the cost components:

Pre-release Components

- The percentage of bugs detected and fixed pre-release
- The average cost per bug fix pre-release

Post-Release Component

- The percentage of security bugs believed to be discovered and exploited by attackers (this number may be a guess on the part of the user, so it is best to try running the program with a series of guesses to build up a range of values).
- Public relations costs, including the amount of effort expended in terms of man month plus any additional cost incurred.
- Legal costs, including the amount of effort expended in terms of man month plus any additional cost incurred.
- Client support costs in term of man month expended.
- The effort on future sales revenue lost due to a security breach. Sales are assumed to recover after one year.
- Additional incidental costs in dollars.
- He overall cost involved in diagnosing a problem post-release in man month plus incidentals.
- The overall cost involved in patching software post-release in man month plus incidental.
- The overall cost involved in software testing post-release in man months plus incidentals.
- The user's average cost per man month.

Benefit Analysis

There are two separate equations used in the benefit analysis. The *Expected Cost Pre-Security (ECP_{re}),* which calculates the expected losses before security is added, and the *Expected Cost Post-Security (ECP_{ost}),* which calculates the expected costs after security is embedded into the software. The equations are as follows:

Equation 1

$$\text{Expected Cost Pre-Security } (ECP_{re}) = (Pre_{Rcom} + Ex_{comp} + Post_{Rcomp}) * N_{Obug} * P_{size}$$

$$Pre_{Rcom} = (\%_{DPpreR} * Pre_{Rfix\text{-}cost})$$

$$Ex_{comp} = ((Num_{bugs} / (Num_{bugs} + NumN_{bugs})) * (1 - \%_{DPpreR}) * (\%_{bugexp} * T_{cost}))$$

$$Post_{Rcomp} = (1 - \%_{DPostR}) * Post_{RT})$$

$$N_{Obug} = Sec_{bug} + Non_{Secbug}$$

$$P_{size} = No_{LineC} / 1000$$

$$Post_{RT} = T_{DCost} + T_{PCost} + T_{TCost}$$

$$T_{cost} = T_{PRcost} + T_{lcost} + T_{clientScost} + L_{prof} + O_{cost}$$

$$L_{prof} = (\%_{Slost} * T_{salesG}) * P_{margin}$$

Where Pre_{Rcom} is the pre-release component, Ex_{comp} is the exploit component, $Post_{Rcomp}$ is the post-release component, N_{Obug} is the number of bugs, P_{size} is the *project size,* $\%_{DPpreR}$ is the percentage of bugs discovered during software pre-release stage, $Pre_{Rfix\text{-}cost}$ is the pre-release fix cost, Ex_{comp} is the exploited components, Num_{bugs} is the number of security bugs, $NumN_{bugs}$ is the number of non security bugs, $\%_{DPpreR}$ is the percentage of bugs discovered during the pre-release stage, $\%_{bugexp}$ is the percentage bugs exploited, T_{cost} is the total costs, T_{PRcost} is the total pre-release costs, T_{lcost} is the total legal cost, $T_{clientScost}$ is the total client support costs,

L_{prof} is the loss profits, O_{cost} is the other costs, $\%_{Slost}$ is the percentage of sales lost, T_{salesG} is the total sales revenue generated, P_{margin} is the profit margin, Sec_{bug} denotes security bugs, Non_{Secbug} denotes non security bugs, No_{LineC} is the number of line of code, $\%_{DPostR}$ is the percentage of bugs discovered during software post-release stage, $Post_{RT}$ is the post release total, T_{DCost} is the total diagnostic cost, T_{PCost} is the total patch and T_{TCost} is the total testing cost.

Equation 2

$$Expected\ Cost\ Post\text{-}Security\ (ECP_{ost}) = (Pre_{Rcom} + Ex_{comp} + Post_{Rcomp}) * N_{Obug} * P_{size}$$

$$Pre_{Rcom} = (\%_{DPpreR} * (1 + Inc_{\%DpreR}) * Pre_{Rfix\text{-}cost})$$

$$Ex_{comp} = ((Num_{bugs} / (Num_{bugs} + NumN_{bugs})) * (1 - \%_{DPpreR})* (\%_{bugexp} * T_{cost}))$$

$$Post_{Rcomp} = (1 - \%_{DPostR})*(1 + Inc_{\%DpreR}) * Post_{RT}$$

$$N_{Obug} = Sec_{bug} + Non_{Secbug}$$

$$P_{size} = No_{LineC} / 1000$$

$$Post_{RT} = T_{DCost} + T_{PCost} + T_{TCost}$$

$$T_{cost} = T_{PRcost} + T_{Icost} + T_{clientScost} + L_{prof} + O_{cost}$$

$$L_{prof} = (\%_{Slost} * T_{salesG}) * P_{margin}$$

Where $Inc_{\%DpreR}$ denotes increase in percentage discovered during the pre-release stage.

Equation 3

$$Total\ Benefit\ (T_{ben}) = Equation\ 1 - Equation\ 2$$

$$ECP_{re} - ECP_{ost}$$

In combination to the cost-benefit analysis implicitly, weighing the total expected costs against the total expected benefits of one or more actions in order to choose the best or more profitable option. The formal process is often referred to as either CBA (Cost-Benefit Analysis) or BCA (Benefit-Cost Analysis). The decisions as to whether to practice secure software development or not could be gotten from the results of the Cost-Benefit Analysis.

Security Costs Analysis

To calculate the cost of secure software development, measuring factors that must be considered when adding security to the software development were posited including major costs that may be incurred. Major cost items involved:

i. Use of new CASE tools or hardware / software that is required for developing secure software. If these tools are generic and can be used with other projects as well, their capital costs should be appropriated appropriately.

ii. User training – security requirements may require the firm to provide the developers some training. The costs are two folds:
    a. there is the direct cost of training (i.e. hiring and paying someone to train employees;
    b. there is an opportunity cost in term of time when employees are undergoing training. Both should be incorporated.

iii. Increase in the effort level (person month *(PM))*) due to security components and cost of increased effort. This is base on the following formulae:

$\Delta E = E(\text{with security}) - E(\text{without security})$

iv. Where E is the effort level (PM) and ΔE is the additional effort required to develop a secure product. Since COCOMO-II has been used extensively in estimating E(without security) and users are quite familiar with such models, one can also estimate E(with security) with some confidence. The formula for effort level $E(\text{in person month})$ is given by:

$$E(\text{estimated}) = L_{code}\ Scale_F \times \Pi(Mul_E)$$

Where L_{code} is lines of code in thousands and $Scale_F$ is a scaling factor. In particular, $Scale_F$ can be set as $Scale_F = 1.01 + 0.01\ SUM(f)$, where f are five scaling factors. Mul_E is effort multipliers (there are 17 of them). Both f and Mul_E are rated on the scale from very low, low nominal, high, very high, and extra high. The weight of each factor can be quantified based on calibration with various projects and continues to evolve.

iv. Impact of delay in product introduction due to additional security. More effort may cause project delay. In software, delay may or may not be costly. Thus to calculate the cost of additional software security, the following information must be obtained:

- An estimate of the percentage change in source code size from adding security protection. For example, it is believed the program's size will grow 5% in number lines of code due to increased security measures.
- The estimated complexity (from very low to very high) of the software project before and after security is added. For example, the staff member may think the software project's complexity was low before adding security measures but moderate or high afterwards.
- An estimate of the level of program documentation (from very allow to very high) required before and after security measures are increased.
- The estimated system analyst capability (overall experience from very low to very high) before and after security is added (i.e. an estimate of the effect that the addition of security will have on the system analysts' technical capabilities.
- The estimated programming team capability (overall development experience from (very low to very high) before and after security is added (i.e. an estimate of the effect that the addition of security measures will have on the programming teams' technical capabilities)
- An estimate of familiarity with tools that are required to add security to the software project. The staff member is asked how he or she feels the development teams' experience with those tools (from very high to very low) before and after addition of security.
- An estimate of the change in development time required (from very high to very low) before and after security measures are added. For example, the staff member may believe that before security was added, the project would take moderate amount time to complete, but after security is added it will take a very high amount of development time.
- An estimate of the overall effort (in man months) required to develop the software before security is added. There may be records indicating that it takes on average 5 man months to complete a similar project.

- The average cost per man month, which is the amount spent on average per 30 days of an employee's time.
- An estimate of reliability requirements (from very low to very high) before and after security. Before adding security, the staff member may have believed that the program only needed to be highly reliable, whereas to meet the claims and safety requirements of a security system it needs to be very highly stable.
- An estimate of the cost for user training. To calculate this, the number of employees being trained, the average time (in days) they spend in training, and the average cost per employee per day is included.
- An estimate of any losses that will be incurred due to a delayed market entry (in any money currency being utilized i.e. naira or dollars)

Formally, we posit four security cost equations. The effort cost *(EF_{cost}),* opportunity cost *(op_{cost}),* the cost of new CASE tools or hardware/software *($Cost_{hs}$)* and the total cost (T_{cost}). The equations are as follows:

Security Cost Equation 1: Effort Cost

$$EF_{cost} = N_{eff} * cost_{ppm}$$
$$N_{eff} = Ol_{eff} * eff_{chg}$$
$$eff_{chg} = ((1 + \%_{CSIn})^{1.15}) * (com_{b4} / com_{af}) * (doc_{b4} / doc_{af}) * (an_{cb4} / an_{caf}) * (p_{cb4} / p_{caf}) * (Tl_{b4} / Tl_{af}) * (T_{b4}) * (R_{b4} / R_{af})$$

EF_{cost} which denotes the effort cost, N_{eff} is the new effort, $cost_{ppm}$ the cost per person month, Ol_{eff} is the old effort, eff_{chg} denotes effort change, $\%_{CSIn}$ denotes percentage code size increase, com_{b4} denotes complexity before, com_{af} denotes complexity after, doc_{b4} denotes documentation before, doc_{af} denotes documentation after, an_{cb4} denoted analysis capability before, an_{caf} denotes analyst capability after, an_{cb4} denotes programmer capability before, an_{caf} denotes programmer capability after, Tl_{b4} denotes tools before, Tl_{af} denotes tools after, T_{b4} denotes time before, R_{b4} *reliability before and* R_{af} reliability after. The numerator and denominator of each of the terms in the "effort change" calculation is derived from the user's answers to the corresponding values stated earlier and is assigned a numerical value.

Security Cost Equation 2: Opportunity Cost

$$op_{cost} = No_{Tem} * Av_{len} * (Av_{emcost} / 365)$$

op_{cost} denotes opportunity cost, No_{Tem} denotes number of employees in training, Av_{len} denotes average length of training and Av_{emcost} denotes average employee cost.

Security Cost Equation 3: Cost of new CASE tools or hardware/software

$$cost_{hs} = cap_{ex}.$$

$cost_{hs}$ denotes cost of new hardware/_software and cap_{ex} denotes capital expense to buy additional hardware and software. So the additional capital expense can be divided into 10 projects equally, thus cost =capital expense / 10

Security Cost Equation 4: Total Cost

$$T_{cost} = EF_{cost} + Tra_{cost} + op_{cost} + cost_{hs} + del_{MCost}$$

T_{cost} denotes total cost, Tra_{cost} denotes training cost and del_{MCost} denotes delay to market cost.

CONCLUSION

Because the IT industry itself is changing and growing over time, certainly the security considerations (potential holes and solutions) are changing right along with it. Security professionals continue to battle against external attackers, cyber terrorists, automated worms and viruses, and unfortunately, unknowing employees who mistakenly grant access to an attacker by falling prey to a social engineering attacks. Software developers and security professionals must therefore work as a team and make the necessary effort to stay abreast of these changes and pay attention to the fact that security should not be scrammed on the software at the later phases of software development life cycle, instead like other aspects of information processing systems; the security process should be a continuous process throughout the software development life cycle. This is crucial, for it will enable software developers protect their customers trust and confidentiality. And by making use of the software security costs and benefits models analyzed in this paper, the cost and benefits of adding security to an insecure software can be evaluated and trade-offs set.

REFERENCES

Arora A., Frank S, and Telang R. (2008). Estimating Benefits from Investing in Secure Software Development, Building Security \in, Carnegie Mellon University.

Alcorn B. (2006). Developing Secure Software, Blackboard Inc, Washington Marriott pp 1 – 23.

McGraw, G. "Software Security". IEEE Security & Privacy, 2(2), 2004, pp.80-83.

Egwali A. O. and Akwukwuma V.N. (2008): Security Framework for Software Process Models: Measures for Establishing a Choice. *Asian journal of Information Technology,* 7 (11): 463 - 471.

Kumar S. H. (2006). Seminar Report on The Study of Viruses and Worms. KReSIT.1.1.T Bombay, pp 2 – 6.

Kabay, M. E. (2008). A Brief History of Computer Crime: An Introduction for Students. Norwich University, pp. 4 – 56.

Davis, A. M., Bersoff E. H. and Comer, E. R. (1988): A strategy for comparing alternative software development life cycle models. Software Engineering, IEEE Transactions on, Vol. 14, No. 10. pp. 1453-1461.

Gordon L. A. (2003). Economic Aspect of Information Security. Available online at: http://www.umiacs.umd.edu/docs/umiacspresentation.pdf

Cavusoglu H. (2006). Economics of Security Patch Management. Available online at: http://www.weis2006.econinfosec.org/docs/5.pdf

August and Tunca (2006). Let the Prates Patch? An Economic Analysis of Software Security Patch Restrictions. Available online at: http://www.mansci.journal.infors.org/../170

Pietersz G. (2009). Net Present Value. Available online at: http://www.sunbeltsoftwrae.com/spam.mht