How Can Secure Software be Trusted?

Lynn Futcher[1], Rossouw von Solms[2]

Nelson Mandela Metropolitan University, Port Elizabeth, South Africa,

Abstract

The security of software applications is a major concern, especially for information owners, software developers and users. Increasingly, these stakeholders need to be confident that the software applications being developed are secure and can be trusted when used in the intended environment. However, a problem exists in terms of how to confidently address the security of software applications in order to protect the information to be stored, processed and transmitted by them, thereby increasing their associated levels of trust. The purpose of this paper is therefore to address some key aspects relating to the security and trustworthiness of a software application functioning within the intended environment. These key aspects include those relating to the security controls implemented and installed by the software developers and those involving the actual usage of the security controls implemented.

CATEGORIES AND SUBJECT DESCRIPTORS

D.2.1. and K.6.5.

GENERAL TERMS

Security

KEYWORDS

Secure software, trusted software, information security, secure software development, common criteria.

1. INTRODUCTION

The Internet has created a fundamental and radical change in the role that software plays in the world today. No longer does software simply support back offices and home entertainment - it has become deeply intertwined in our everyday lives and is regarded as the lifeblood of many organisations [1]. In the current Internet-dominent era, virtually all computers (including servers, desktop personal computers, cellular phones and other mobile devices) are interconnected. In this way software today provides immediate, global access to information, it enables electronic commerce and it automates supply chains. It has also become an integral part of our household appliances, cars and home security systems. The problem is that these interconnected computers and networks can be attacked at various points, putting the associated information at risk. A substantial portion of these attacks on systems occur through exploiting vulnerabilities in the software that forms an integral part of the system. This raises the question of *'Why do these vulnerabilities exist in software?'*.

Unfortunately software often is developed with minimal concern for security. According to Viega and McGraw [1], this could possibly be attributed to the demanding constraints of project management, including time, cost and resources. In today's economic environment, these are still primary contributing factors. In addition, security goals are often believed to clash directly with many of the goals of modern software development methodologies which tend to pay specific attention to the needs, wants, and limitations of end-users. The goals of users may include functionality, usability, efficiency and simplicity. Many software developers do not possess the knowledge and expertise necessary to cater to the security goals of an application. Furthermore, Howard and LeBlanc [2] suggest that security is boring and is often seen as a functionality disabler. For many developers it means not being able to do something new and exciting. This, together with the fact that security is difficult to measure, means that security aspects are often neglected during the development process. This eventually leads to vulnerabilities in the software product.

It is evident that for many organisations, security is still considered as something that 'gets in the way' and costs money, while offering little or no financial return. However, there are many arguments supporting the development of secure software. First and foremost, secure products suggest quality in terms of confidentiality (protection from disclosure), integrity (protection from alteration) and availability (protection from destruction) [3]. The failure to design and build secure software, from the perspective of the software developer, leads to more work in the long run and a bad reputation for the developers, users, company and company's clients. This, in turn, can lead to the loss of sales for an organisation as customers switch to a competing product perceived to have

[1] Lynn.Futcher@nmmu.ac.za

[2] Rossouw.vonSolms@nmmu.ac.za

better security support. Users, on the other hand, do not want their systems to be infected by viruses, their credit card information or their personal data to be compromised. Software applications are therefore expected to securely process, transmit and store sensitive user and corporate information. Today, it can be rightly argued that users and information owners are demanding more secure software applications and now consider such systems as a right and not a privilege. Software systems must therefore be trusted to process, store and transmit all related sensitive information in a secure manner. However, it is apparent that a problem exists in terms of how to confidently address the security and trustworthiness of software applications thereby meeting the needs of all the stakeholders, including the information owners, software developers and users.

The purpose of this paper is to address some key aspects related to the security and trustworthiness of a software application functioning within a specific environment. Section 2 introduces the notion of secure and trusted software, while Section 3 focusses on secure software development by referring to software development and information security standards and best practices. Section 4 takes a closer look at existing evaluation frameworks and criteria that provide a conceptual grounding for the key aspects proposed for secure and trusted software. These key aspects are described in Section 5.

2. SECURE AND TRUSTED SOFTWARE

Attacks on software have increased dramatically since the 1980s. Traditional perimeter defenses such as firewalls, intrusion detection and anti-virus systems are no longer able to stop these software attacks as hackers increasingly focus on the software layer [4]. Security has therefore become an essential requirement for software developers. However, since traditional software development methodologies do not pay much attention to security aspects, addressing software security problems effectively is often difficult. The majority of security weaknesses exploited by viruses, worms and other malware can be attributed to poor software design [1,5]. These weaknesses, however, are not intentionally introduced by software developers.

It is apparent that there is far more to software security than avoiding the often discussed problem of buffer overflows. According to Pfleeger [6], secure software applications need to be correct, complete and exact. A software application is correct if it meets the requirements for which it was designed; complete if it meets all the specified requirements; and exact if it performs only those operations specified by the requirements [6]. McGraw [7] defines software security as '*the idea of engineering software so that it continues to function correctly under malicious attack'*. Viega and McGraw [1] further state that the problem of security is relative and that there is no such thing as 100% security. In addition, they suggest that software security can be seen as a measurement of how robust a specific software application is with respect to a particular security policy and that auditing is an essential part of software security.

The Software Assurance Forum, established jointly by the United States Department of Homeland Security and Department of Defence, defines secure software as that which exhibits the properties of dependability, trustworthiness, resilience and conformance [8]. These are stated as software assurance objectives. While dependability refers to the level of confidence that the software, when executed, will function only as intended, trustworthiness depends on the extent to which no exploitable vulnerabilities or malicious logic exist in the software. Resilience, on the other hand, relates to the ability of the software to recover quickly to an acceptable level of operation, if compromised in any way. Conformance requires that the software conforms not only to the requirements specified but also to relevant standards and procedures [8].

From the literature studied, it is evident that a consistent approach to providing secure and trustworthy software is needed. Addressing the security and trustworthiness of software applications is difficult since the traditional software development lifecycle (SDLC) does not particularly take security into consideration. In the past, software developers generally focused on core functionality and features. Security was typically only addressed as an afterthought and in a very ad hoc manner [9,10]. The following section addresses secure software development by referring to various software development and information security standards and best practices.

3. SECURE SOFTWARE DEVELOPMENT

Many organisations produce software development and information security standards. The Institute for Electrical and Electronic Engineers (IEEE), the National Institute for Standards and Technology (NIST), the International Standards Organisation (ISO), the American National Standards Institute (ANSI), the American Department of Defense (DoD), the British Standards Institute (BSI), the Common Request Object Broker Architecture (CORBA) and the Object Management Group (OMG) are all well known sources of such standards.

Secure software is a software development problem [4]. The idea of integrating security into the software development lifecycle has been widely addressed. This stems from the fact that security cannot be added on as an afterthought, but must be considered from the outset. According to NIST [11], integrating information security requirements into the SDLC is the most efficient and cost-effective method of ensuring that the organization's protection strategy is reflected in the information systems needed to support the processes of the organization. Security needs to be considered throughout the software development lifecycle regardless of which methodology is followed.

3.1 Software Development Standards and Best Practices

The IEEE regularly publishes software development standards and ANSI works closely with the IEEE in developing industrial software development standards. Similarly, the DoD publishes military standards for software and the BSI serves as a rich source of standards concerning every aspect of software development. ISO standards cover design and description in ISO 6593; documentation in ISO 9127; and software quality management in the ISO 9000 series. ISO/IEC (International Electrotechnical Commission) JTC1 SC7 is responsible for the standards related to software quality and software engineering.

Software developers tend to follow various development methodologies. These range from the classic waterfall model, to Boehm's spiral model, Capability Maturity Model Integration (CMMI), the Team Software Process (TSP) and Personal Software Process (PSP) to the more recently adopted agile methods and Comprehensive Lightweight Application Security Process (CLASP). However, there is little evidence that any of these methods create more secure software [12]. According to Howard and Lipner [12], the primary difference between Microsoft's Secure Development Lifecycle (SDL) and CMMI, TSP and PSP is that whereas SDL focuses solely on security and privacy, CMMI, TSP and PSP are mainly concerned with

improving the quality and consistency of development processes in general. CMMI, TSP and PSP neglect to make any specific provisions or accommodations for security [12].

More recently, agile development methods have become popular for developing software. These methods attempt to reduce the overall risk of software development projects by building software in very rapid and short iterations, called sprints or timeboxes. Although the Microsoft Solutions Framework (MSF) for Agile Software Development adds some security checklists and threat modeling, it is very superficial and focuses only on some basic programming practices for security. Howard and Lipner [12] claim that there is no reason why SDL cannot be adopted by agile methods of software development. However, according to Goerzel [8], just because software performs information security-related functions, it does not mean that the software itself is secure. The functionality provided by a software application needs to be measurable, observable and testable [13].

It is evident that few software development methodologies actually cater for security and those that do state mostly 'what' must be addressed and not 'how'. Thus, a lot is left to the software developer to interpret and to determine. In addition, few real guidelines exist to provide assistance as to what is correct and what is enough when integrating security into software applications.

3.2 Information Security Standards and Best Practices

The National Institute of Standards and Technology (NIST) is a measurement standards laboratory which is a non-regulatory agency of the United States Department of Commerce. NIST publications address specific security considerations and therefore provide an excellent source of security standards and best practices. For example, risk assessment is addressed in NIST SP 800-30 *'Risk Management Guide for Information Technology Systems'* [14] and security in the SDLC is addressed in NIST SP 800-64 *'Security Considerations in the Information System Development Life [11].*

According to NIST [11], regardless of the type of software development methodology used by an organisation, information security must be integrated into the SDLC from the earliest stages to ensure appropriate protection of the information to be transmitted, processed, and stored by the system. In addition, NIST promotes risk management as playing a critical role in protecting an organisation's information assets from IT-related risk [15]. NIST therefore supports an integrated approach to secure software development that specifically addresses risk management and in so doing enables security to be planned, built in and deployed as an integral part of the development process.

However, articulation of the desired system security properties is essential to integrating security into the SDLC. These system security properties are commonly referred to as 'security requirements' [16]. Although there are many ways to express these requirements, NIST [16] refers to using the concepts described in the Common Criteria for Information Technology Security Evaluation [17], ISO/IEC 15408, also known as the CC. In addition to articulating the security requirements of a system, NIST [16] further states that '*the correct and effective use of information security controls is a fundamental building block of information security'*. However, a certain level of assurance is required to provide confidence that the security controls identified will operate correctly and effectively in the intended operational environment.

The CC is useful in that it not only provides a standard vocabulary and format for stating the security requirements of a system. It also provides various levels of assurance of a product or information system that is to be trusted [16]. The following section provides further discussion of the CC and it's relevance in developing trusted and secure software applications.

4. TRUSTED SECURITY EVALUATION CRITERIA

The Common Criteria [17] is currently the international standard (ISO/IEC 15408) for computer security and has superceded the Trusted Computer Security Evaluation Criteria (TCSEC) and the Information Technology Security Evaluation Criteria (ITSEC). The purpose of the CC is to allow software developers to specify their security requirements, to specify the security attributes of their products, and to allow evaluators to determine if the products actually meet the security requirements initially identified. According to the CC, these security requirements are categorised according to functional and assurance requirements. Functional requirements define the desired security behaviour and can be equated to how strenuously the security controls actually perform their function in the intended environment. Assurance requirements, on the other hand, are the basis for gaining confidence that the chosen security controls are effective and correctly implemented.

The CC [17] uses specific terminology for evaluation which needs to be understood. The *Target of Evaluation* (TOE) is basically the product or system to be evaluated. For the purposes of this paper, the TOE refers to the particular software application being developed. The *Protection Profile* (PP) is determined by the specific security controls that form part of the TOE. The PP needs to be evaluated at a desired assurance level. In this way the PP (that constitutes all the security controls and mechanisms) is supposed to provide the assurance of the TOE. The *Security Target* (ST) is a specification of the security requirements of a TOE and is to be used as a baseline for evaluation. This begins by describing the assets and the potential threats to those assets. In essence, this is the process of some risk assessment. Having identified the specific risks to the TOE, the ST then sets the benchmark for the security controls and mechanisms required to respond to and counter these risks. The aim of the evaluation process is to determine whether the PP of the TOE meets the ST of the TOE.

Figure 1 clearly illustrates some of the key *security* concepts and relationships used throughout the CC [17]. The purpose of security is to protect assets. An asset is an entity which someone (owner, user, threat agent) places a value upon. From a software development perspective, assets are typically in the form of information that can be stored, processed and transmitted via software applications. It is therefore important that software applications meet the security requirements laid down by the owners of the assets implicated. Since threat agents (hackers, malicious users) may also place value on the assets, they may wish to abuse and/or damage the assets thereby leading to a loss of confidentiality, loss of integrity or loss of availability. These threats therefore increase the risk to the information assets which require protection by the imposing of controls (also referred to as security mechanisms, countermeasures or safeguards). It is necessary to note that the risks to be managed take on different levels of urgency and importance in different situations. For this reason it is essential that the ST of a software application be well defined, taking the particular threats into account. For example, the ST for an electronic

banking application and an electronic personal diary system will differ significantly.

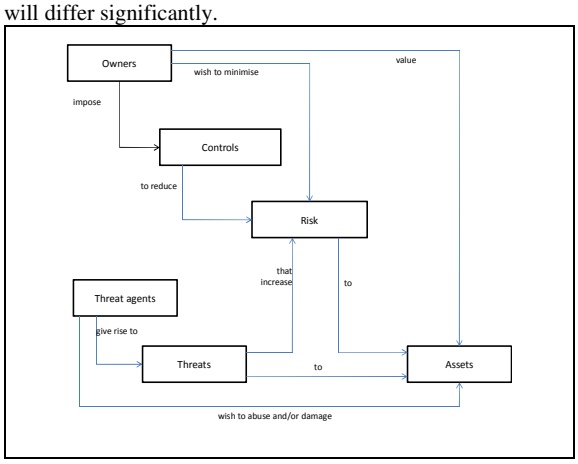

Figure 1: Key Security Concepts and Relationships [17]

As already stated, the purpose of the CC is to allow software developers to specify their security requirements, to specify the security attributes of their products, and to allow evaluators to determine if the products actually meet the security requirements initially identified. Figure 2 therefore illustrates some of the key *evaluation* concepts and relationships used throughout the CC [17]. According to the CC, some form of evaluation is required to provide confidence to the owners of the information assets that the controls implemented are both sufficient and correct thereby minimising the risk to these information assets. These risks may be related to a loss of confidentiality, loss of integrity or loss of availability of the information assets concerned. The sufficiency of controls is determined by the ST which clearly describes the assets implicated together with their potential risks, whereas the correctness of controls may be impacted by poor design and inadequate implementation and testing. A problem exists in that most information asset owners lack the know-how, expertise and resources to determine the sufficiency and correctness of the security controls implemented in software. They therefore call for an evaluation in order to increase their confidence with the software application in question [17].

From an assurance point of view, it is important that the PP is tested for effectiveness and correctness. *Effectiveness* refers to whether the security controls that form part of the PP are sufficient enough to respond to the risks identified in the ST. Effectiveness therefore requires that the assets and threats associated with the TOE be considered. *Correctness* is determined by whether the security controls stated in the PP are correctly implemented and installed [17].

Some of the drawbacks of the CC include the cost and duration of performing formal evaluations [18]. In line with the underlying principles of the CC, the following section addresses some key aspects relating to the security and trustworthiness of a software application to assist the various stakeholders in the formal assessment of software applications.

5. KEY ASPECTS OF SECURE TRUSTED SOFTWARE

Software trustworthiness may be defined as 'the level of confidence that a software application will fulfill the given set of goals and requirements while remaining free from threats and maintaining normal operation under all possible circumstances' [19]. Trusted software therefore does what it is designed to do

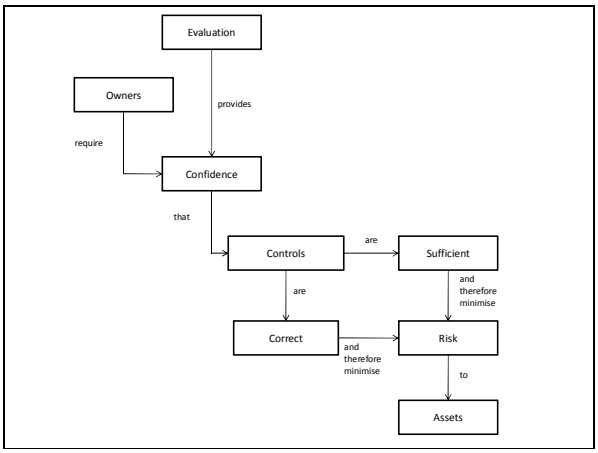

Figure 2: **Key Evaluation Concepts and Relationships [17]**

accurately and reliably. In addition, it should not allow anything other than that which was designed as explicitly allowable. In order for a software application to be trusted and deemed secure, the following questions need to be satisfactorily answered:

- Do the security controls function properly and are they sufficiently strong in the given context?
- Are the security controls adequate, taking the related threats into account?
- Are the security controls properly implemented and installed to provide optimal operation and maximum protection?
- Is correct user behaviour enforced?

These questions may be answered by addressing some key aspects related to the security and trustworthiness of software applications namely functionality, effectiveness, correctness and usage.

From a functional point of view this means that the chosen security controls must perform as expected when applied in the intended environment. In addition they must be strong enough to address the risk associated with the information assets implicated. The **functionality** of a software application may therefore be answered via the question 'Do the security controls function properly and are they sufficiently strong in the given context? '.

The effectiveness of security controls relates to the extent to which they do what they are supposed to do taking the envisaged threat environment into account. According to Howard and LeBlanc [2], the need for security and its strength are context-driven. This means that different situations call for different solutions. The risks to be managed take on different levels of urgency and importance in different situations. The key when developing secure software products is to design and build them so that they are sufficiently secure for the environment in which they will operate. A system that is secure in one context may be completely insecure when placed in another. The **effectiveness** of a secure software application may be answered through the question 'Are the security controls adequate, taking the related threats into account? '.

Correctness relates to the correct implementation and installation of the chosen security controls. It requires that this is carried out in accordance with accepted standards and best

practices. Adherence to secure coding best practices and carrying out security testing also address the correctness of security controls. **Correctness** is best answered by asking 'Are the security controls properly implemented and installed to provide optimal operation and maximum protection? '.

It is argued that the usage of software is just as important as the correctness of software itself [20]. Users are increasingly required to use software applications in a safe and secure way. One way to support the correct use of software is to provide ongoing security training and education. Microsoft recommends that tools and guidance should accompany software to support secure and trusted usage. In addition, users need to be encouraged to accept responsibility and become accountable for their behaviour and actions. The question of **usage** may therefore be answered by asking the question 'Is correct user behaviour enforced?' draft

Figure 3, based on some of the key security and evaluation concepts addressed in Figures 1 and 2, has been drafted by the authors to clearly illustrate these important aspects of secure and trusted software. This diagram shows that various stakeholders call for trust in software. Firstly, information owners would like to trust that their information will be protected by the software applications that are responsible for its storage, processing and transmission - thereby ensuring its confidentiality, integrity and availability. Furthermore, software developers would like to trust that the software they develop will be used as intended by the users of their applications, and users expect that the software they use will carry out their tasks securely.

However, software applications tend to function in a specific environment. Unfortunately, these environments are exposed to risk since threat agents tend to operate within these environments. This gives rise to threats which exploit existing vulnerabilities in order to compromise the highly valued information assets. This increases the risk associated with them as the specific environment becomes exposed to the associated risk. In order to reduce the risk associated with the information assets implicated, software developers implement controls. These controls, however, need to be functional, effective and correct according to the environment in which they will operate.

From a security point of view, the security controls implemented need to function appropriately, be suitably strong, and be safe to use. The effectiveness of a security control, however, can only be determined by considering the specific environment in which it functions, taking the associated risk into account. It may therefore be necessary to identify numerous security controls in order to reduce this risk. This means that although a control may be functional in its own right, it may not be sufficient to mitigate the risk associated with the given environment. Having determined the necessary security controls, the correctness of these controls may be achieved through accurate and correct implementation and installation. This helps ensure the protection of the information assets implicated. However, full protection can only be achieved by the users of the software since they are ultimately responsible for using the software in a secure and responsible manner. It is therefore argued that responsibly usage is a further key aspect to consider when discussing trusted software.

By taking into account the key aspects of functionality, effectiveness, correctness and usage, it is evident that every stage of the software development plays a vital role in ensuring the security of software applications. For example, if a risk analysis is not carried out in the beginning of the SDLC, then effectiveness might suffer since the specific threat environment would not have been considered. If some security mechanisms or controls are self developed and the ST is quite 'stringent', the functionality might suffer as the mechanisms may not be strong enough to withstand attacks. Likewise, if functionally sound security mechanisms are chosen, but poorly implemented into the software or poorly installed, then correctness might suffer and if the software is securely developed and the users do not operate it in a secure manner, then usage will suffer.

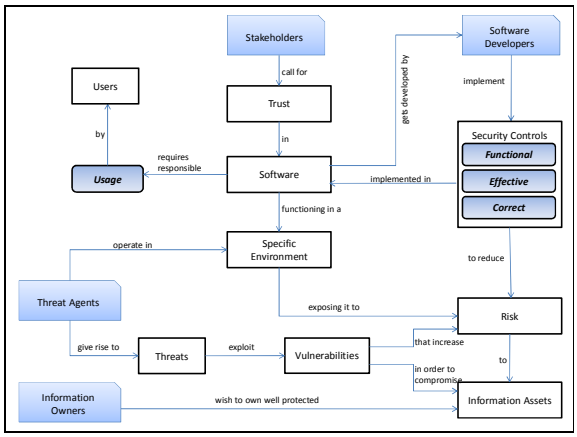

Figure 3: Key Aspects of Secure Trusted Software

6. CONCLUSION

Based on the conceptual foundation provided by the CC, this paper proposes *functionality*, *effectiveness*, *correctness* and *usage* as fundamental aspects for secure and trusted software applications. By considering these key aspects, a higher level of security and trust could be provided for all stakeholders including the information owners, software developers and users of the software. Any methodology for secure software development should therefore take into consideration these key aspects of secure and trusted software as described in Section 5. Although various aspects of secure software development have been extensively researched, the integration of these aspects is still lacking. For example, much of the literature studied refers to the importance of defining security requirements, but this only addresses the *functionality* of a software application and not the other three elements. Many other sources focus specifically on risk analysis and threat modeling, which impacts the *effectiveness* of the system while others centre on secure software principles and best practices or secure coding and testing which all relate to *correctness.* Although the *usage* of software applications is discussed in some organizational policies and procedures, most software development methodologies pay very little attention to this aspect. This paper combines these aspects as fundamental to ensuring secure and trusted software.

REFERENCES

[1] Viega, J., & McGraw, G. (2001). *Building Secure Software.* Addison-Wesley.

[2] Howard, M., & LeBlanc, D. (2003). Writing secure code : Practical strategies and techniques for secure application coding in a networked world. Microsoft Press.

[3] Paul, M. (2008a). *Software Security: Being Secure in an Insecure World.* Retrieved September 1, 2009, from The International Information Systems Security Certification Consortium Inc (ISC2): http://www.isc2.org/uploadedFiles/(ISC)2_Public_Content

/Certification_Programs/CSSLP/CSSLP_WhitePaper_3B.pdf

[4] Taylor, J., & Reinstein, M. (n.d.). *Application Security by Design: Security as a Complete Lifecycle Activity.* Retrieved March 8, 2010, from Security Innovation: www.securityinnovation.com

[5] Ghosh, A. K., Howell, C., & Whittaker, J. A. (2002, January/February). Building Software Securely from thew Ground Up. *IEEE Software* , 14-16.

[6] Pfleeger, C. P. (2007). *Computer Security.* Retrieved November 3, 2009, from AccessScience@McGraw-Hill: http://www.accessscience.com/

[7] McGraw, G. (2004). Software Security. IEEE Security and Privacy , 80-83.

[8] Goertzel, K. (2009, January 09). Introduction to Software Security. Retrieved December 3, 2009, from Build Security In: https://buildsecurityin.us-cert.gov/daisy/bsi/547-BSI.html

[9] Viega, J. (2004, October 15). *Security in the Software Development Lifecycle.* Retrieved April 13, 2010, from IBM: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/oct04/viega/

[10] Paul, M. (2008b). *The Need for Secure Software.* Retrieved September 1, 2009, from The International Information Systems Security Certification Consortium Inc (ISC2): http://www.isc2.org/uploadedFiles/(ISC)2_Public_Content/Certification_Programs/CSSLP/CSSLP_WhitePaper.pdf

[11] NIST. (2008a, October). *SP 800-64: Security Considerations in the Systems Development Life Cycle.* Retrieved October 14, 2009, from National Institute of Standards and Technology: Special Publications (800 Series): http://csrc.nist.gov/publications/nistpubs/800-64-Rev2/SP800-64-Revision2.pdf

[12] Howard, M., & Lipner, S. (2006). *The Security Development Lifecycle.* Washington: Microsoft Press.

[13] SAFECODE. (2008, February). *Software Assurance: An Overview of Industry Best Practices.* Retrieved December 7, 2009, from SAFECODE: Software Assurance Forum for Excellence in Code: http://www.safecode.org/publications/safecode_bestpractices0208.pdf

[14] NIST. (2002, July). *SP 800-30: Risk Management Guide for Information Technology Systems.* Retrieved October 14, 2009, from National Institute of Standards and Technology: Special Publications (800 Series): http://csrc.nist.gov/publications/PubsSPs/sp800-30.pdf.html

[15] NIST. (2008b, April). Managing Risk from Information Systems: An Organisational Perspective. Retrieved December 3, 2009, from National Institute of Standards and Technology : csrc.nist.gov/publications/drafts/800-39/SP800-39-spd-sz.pdf

[16] Stoneburner, G., Goguen, A., & Feringa, A. (2002, July). NIST Special Publication 800-30: Risk Management Guide for Information Technology Systems. Retrieved September 3, 2009, from National Institute of Standards and Technology (NIST): http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf

[17] Common Criteria. (2009, July). Common Criteria for Information Technology Security Evaluation. Retrieved December 1, 2009, from Common Criteria: http://www.commoncriteriaportal.org/files/ccfiles/ccpart1v3.1r3.pdf

[18] Oracle. (2001, July). Computer Security Criteria: Security Evaluations and Assessment. Redwood Shores, California, USA.

[19] Tan, T., He, M., Yang, Y., Wang, Q., & Li, M. (2008). An Analysis to Understand Software Trustworthiness. The 9th International Conference for Young Computer Scientists (pp. 2366-2371). IEEE Computer Society.

[20] Yee, K.-P. (n.d.). User Interaction Design for Secure Systems. Retrieved December 7, 2009, from http://people.ischool.berkeley.edu/~ping/sid/uidss.pdf