International Journal of Intelligent Information Systems
2014; 3(5): 45-54
Published online November 20, 2014 (http://www.sciencepublishinggroup.com/j/ijiis)
doi: 10.11648/j.ijiis.20140305.11
ISSN: 2328-7675 (Print); ISSN: 2328-7683 (Online)

Performance evaluation and operation of enterprise resource planning (ERP) software security system

Diponkar Paul[1], Md. Rafel Mridha[2], Md. Rashedul Hasan[2]

[1]Department of EEE, Prime University, Mirput-1, Dhaka, Bangladesh
[2]World University of Bangladesh, Dhanmondi, Dhaka, Bangladesh

Email address:
dipo0001@ntu.edu.sg (D. Paul)

To cite this article:
Diponkar Paul, Md. Rafel Mridha, Md. Rashedul Hasan. Performance Evaluation and Operation of Enterprise Resource Planning (ERP) Software Security System. *International Journal of Intelligent Information Systems.* Vol. 3, No. 5, 2014, pp. 45-54.
doi: 10.11648/j.ijiis.20140305.11

Abstract: The criteria for selecting the specific systems are - containment of most common sources for attacks, knowledge of the exact location of each security hole, accessibility to the source code and selection of a typical web application such as a human resource management. We followed the human resource (recruiting and working procedure) to integrate all the facilities in a single programmable platform. The applied framework has been used to map a commercial security library to the target mobile application SoC (System-of-Chip). The applicability of our framework to software architecture has been explored in other multiprocessor scenarios. ERP software (or enterprise resource planning software) is an integrated system used by businesses to combine, organize and maintain the data necessary for operations. The fundamental advantage of ERP is that integrating the myriad processes by which businesses operate saves time and expenses. The whole process has been automated using a methodology that extracts the risk of ERP system by analyzing the class diagram of the system. ERP for the business to develop innovative services for new and existing organizations, has achieved operational excellence with streamlined logistics and manufacturing improve financial performance with tighter internal controls and insights connect headquarters, subsidiaries and partners in a single network. Any type of small and large organization who to maintain their work flow in an organized way and having an intensity of clear book keeping like as business & educational institutions as well as social organizations.

Keywords: ERP, SecureCL, Trigger, Cursor etc

1. Introduction

Short for enterprise resource planning, ERP is an organization's management system which uses a software application to incorporate all facets of the business, and automate and facilitate the flow of data between critical back-office functions, which may include financing, distribution, accounting, inventory management, sales, marketing, planning, human resources, manufacturing, and other operating units. ERP software, in turn, is designed to improve both external customer relationships and internal collaborations by automating tasks and activities that streamline work processes, shorten business process cycles, and increase user productivity. A method for standardized processing, an ERP software application can both store and recall information when it is required in a real-time environment. Companies often seek out ERP software systems to pinpoint and mend inefficiencies in a business process or when a number of complex issues exist in the business environment. ERP software systems are also implemented to enhance operational efficiencies, achieve financial goals, manage and streamline the company's operational processes, replace an existing ERP software system that is out of date or unable to handle a company's daily activities; or improve information management through better data accessibility, decreased data reduplication and optimal forecasting features [1]. Many business owners see ERP software systems to be critical to their business functions, as they allow companies to achieve absolute business process automation. While most companies use countless processes, activities and systems to run operations, workflows and procedures can go awry when it comes to today's highly competitive marketplace, thus hindering productivity, growth and profitability. As a result, the implementation of an ERP software application can result in increased productivity, reduced operating expenses,

improved data flow, and optimal performance management. ERP software comes in many forms, including supply chain management, manufacturing, distribution, warehouse management, retail management, and point-of-sale software. ERP software (or enterprise resource planning software) is an integrated system used by businesses to combine, organize and maintain the data necessary for operations. ERP systems merge each of the company's key operations, including the manufacturing, distribution, financial, human resources and customer relations departments, into one software system. For many companies, the ERP software is the heart of their operations and the backbone of the organization. ERP software consists of many enterprise software modules that are individually purchased, based on what best meets the specific needs and technical capabilities of the organization. Each ERP module is focused on one area of business processes, such as product development or marketing. Some of the more common ERP modules include those for product planning, material purchasing, inventory control, distribution, accounting, marketing, finance and HR. As the ERP methodology has become more popular, applications have emerged to help business managers implement ERP in other business activities and may also incorporate modules for CRM and business intelligence and present them as a single unified package [2]. Configuring an ERP system is largely a matter of balancing the way the customer wants the system to work with the way it was designed to work. ERP systems typically build many changeable parameters that modify system operation. Data migration is the process of moving/copying and restructuring data from an existing system to the ERP system [3]. Migration is critical to implementation success and requires significant planning. Unfortunately, since migration is one of the final activities before the production phase, it often receives insufficient attention. Advantages: The fundamental advantage of ERP is that integrating the myriad processes by which businesses operate saves time and expense. Revenue and salary tracking, from invoice through cash receipt. They provide a comprehensive enterprise view (no "islands of information"). They make real–time information available to management anywhere, any time to make proper decisions. This article addresses the latter; rather than propose any new security architecture, we present a security characterization framework [4]. Our approach concerns the security functions of software components by exposing their required and ensured security properties. Through a compositional security contract between participating components, system integrators can reason about the security effect of one component on another. A CSC is based on the degree of conformity between the required security properties of one component and the ensured security properties of another. At the application level, such consent based trust perhaps works fine. But in a component-based development environment, universally shallow commitment regarding component security is dangerously illusive and can trigger costly consequences. Trust requirements in a development environment significantly differ from those of application users. Component security— based on various nondeterministic elements such as the use domain, magnitude of the hostility in the use context, value of the data, and other related factors—is relative, particularly in a component-based development environment. Therefore, software engineers must be assured with more than just a component security or insecurity claim[10]. Whatever small role a component plays, the software engineer cannot rule out its possible security threats to the entire application. Component developers might not be aware of the security requirements of their products' potential operational contexts. Software engineers do not expect such knowledge from the component developer, but they do expect a clear specification of the component security requirements and assurances. 1 This information should be made available if queried at runtime. Developers must be able to do runtime tests with candidate components to find possible security matches and mismatches. The major concern—the disclosure of components' security properties and security mismatches of those properties—has received little attention from the security and software engineering research communities. Current practices and research for security of component-based software consists of several defensive lines such as firewalls, trusted operating systems, security wrappers, secure servers, and so on. Some significant work on component testing, component assurances and security certification has been done, particularly in the last two years. These efforts basically concentrated on how to make a component secure, how to assure security using digital certification, and how to maximize testing efforts to increase the quality of individual components. Undoubtedly, such work is important to inspire trust, but we must explore other possibilities that would let software engineers know and evaluate the actual security properties of a component for specific applications. If the developer doesn't know these attributes during system integration, the component might not be trustworthy [11]. In current practice, the trust-related attributes are often neither expressed nor communicated. Software developers are reluctant to trust a third-party software component that does not tell much about its security profile. Despite these shortcomings, software engineers are still inclined to use them to minimize development effort and time. Today, trust in an application system is based on consent—that is, the user is explicitly asked to consent or decline to use a system. At the application level, such consent-based trust perhaps works fine. But in a component-based development environment, universally shallow commitment regarding component security is dangerously illusive and can trigger costly consequences. Trust requirements in a development environment significantly differ from those of application users. Component security— based on various nondeterministic elements such as the use domain, magnitude of the hostility in the use context, value of the data, and other related factors—is relative, particularly in a component-based development environment. Therefore, software engineers must be assured with more than just a component security or insecurity claim. Whatever small role

a component plays, the software engineer cannot rule out its possible security threats to the entire application. Component developers might not be aware of the security requirements of their products' potential operational contexts. Software engineers do not expect such knowledge from the component developer, but they do expect a clear specification of the component security requirements and assurances. These efforts basically concentrated on how to make a component secure, how to assure security using digital certification, and how to maximize testing efforts to increase the quality of individual components. Undoubtedly, such work is important to inspire trust, but we must explore other possibilities that would let software engineers know and evaluate the actual security properties of a component for specific applications. Since 1999, several seminal books have helped define the software security field. These books introduced the approach to building security in, which practitioners have since enhanced, expanded, and published in various technical articles; including the Building Security In series (see the sidebar).The core philosophy underlying this approach is that security, like dependability and reliability, can't be added onto a system after the fact through the addition of sets of features, nor can it be tested into a sys- tem. Instead, security must be designed and built into a system from the ground up. More than 90 percent of reported security incidents are the result of exploits against defects in the designer code of software, according to the CERT Coordination Center (CERT/CC) of the SEI. Although traditional security efforts attempt to retroactively bolt on devices that make it more difficult for those defects to be exploited, such devices simply aren't effective. Standard-issue software development lifecycle models—ranging from the process-heavy Capabilities Maturity Model (CMM) to the lightweight Extreme Programming (XP) approach—are not focused on creating secure systems. They all exhibit serious shortcomings when the goal is to develop systems with a high degree of The only way to develop systems with required functionality and performance that can also withstand malicious attacks is to design and implement them to be secure. Soft- ware security is thus a full lifecycle undertaking in which critical design decisions and trade-offs must be clearly and thoroughly under- stood. In addition, tools for supporting security engineering (for example, source code analysis tools) must be integrated into the software development environment. By treating software security risk explicitly throughout the soft- ware life cycle, we can properly identify and mitigate the consequences of security failure and successful security attack. For each lifecycle activity, a team made up of security analysts and developers must address security goals and incorporate best practices to assure security. In some situations, existing development methods can be used to enhance security [5]. Current research is also creating new methods that developers and analysts can apply as they build software; however, more research and experimentation are required before the goal of security can become a reality [6]. One way of illustrating a lifecycle approach that incorporates security into each basic phase of software development has been intentionally created to be process agnostic. That is, the best practices and methods de- scribed are applicable to any and all development approaches as long as they result in the creation of software artifacts. Given this approach, software development processes as diverse as the waterfall model, Rational Unified Process (RUP), XP, Agile, spiral development, and CMM involve creating a common set of software artifacts (the most common artifact being code). In this way, we can apply software security best practices and their associated knowledge catalogs regardless of exactly which "base" software process is followed. Figure includes best practices (as does Figure A in the sidebar), knowledge, and tools, all organized ac- cording to software artifacts. The Build Security In (BSI) Software Assurance Initiative seeks to alter the way that software is developed so that it's less vulnerable to at- tack by building security in from the start. BSI is a project of the Strategic Initiatives Branch of the DHS's NCSD, which has sponsored the development and collection of software assurance and software security information that will help software developers and architects create secure systems. The effort is managed by Joe Jarzombek, the DHS director for soft- ware assurance. As part of the initiative, a BSI content catalog will be made available as a Web portal in October. This portal is intended for software developers and software development organizations that want in- formation and practical guidance on how to produce secure and reliable software. The catalog is based on the principle that software security is fundamentally a software engineering problem that we must address systematically throughout the software development life cycle. The catalog will contain links to a broad range of information about best practices, tools, and knowledge. Figure identifies aspects of software assurance covered in the catalog[9]. The BSI portal includes information about which tools developers and security analysts can use to detect and/or remove common vulnerabilities. Of particular interest are static analyses tools that help developers look for common security- critical problems in source code. The best current commercial tools support languages such as Java, CLR, C++, C, and PHP (see key BSI5 in the sidebar).Even with deep technical content, a business case is required to convince industry to adopt secure software development best practices and educate consumers about the need for software assurance. Therefore, each documented best practice addresses the business case for use of that practice. In addition, the portal will include overall business case framework dynamic navigation. The extent to which users will find the content accessible as well as useful will determine how this portal impacts real-world development practices and, thus, overall systems security. The BSI team is trying to make the content approachable in several different ways. For example, a soft- ware engineer might use the catalog to determine applicable security guidelines; an architect might use security principles to determine how to design an n-tier application in a secure fashion; and a development team leader might use the information to justify

software assurance techniques to management by building a business case. Because the repository will be structured and designed to evolve as well as support usage by a variety of user types, it will include a dynamic navigation interface. Once practical guidance and reference materials are available forth day-to-day work most development organizations do, the BSI team plans to identify and organize content for practical guidance and reference materials for enterprise-level security concerns. To help ensure that this software assurance initiative is accepted and supported by the community of soft- ware development organizations, the team is seeking involvement from representatives from industry, academia, and government. Toward this goal, working groups to guide the creation of the BSI software assurance portal have been formed. The Software Technical Working Group (STWG) is composed of respected individuals in the technical community whose primary function is to re- view the portal content's technical veracity and identifies future content [7]. Although the portal is currently in a nascent stage, the BSI team welcomes feedback; prior to the site's launch, you can send it to Jan Philpot at the SEI (philpot@sei.cmu.edu). Community involvement and use is crucial to the portal's success, and we look forward to help from the community in improving software security worldwide. To the best of our knowledge, this paper is the first to experimentally examine the resistance of several security patterns to known categories of attacks. The main contribution of this paper is to propose a complete methodology for calculating the risk of STRIDE attacks on a software system composed of security patterns already from its design. Additionally, we make use of a fuzzy risk analysis framework. Using fuzzy terms is more appropriate when examining the design of a system for security. We cannot apply exact numbers due to the lack of exact information about the security of the system. We note here that we make use of nine levels of risk, which leads to better granularity compared to using fewer levels[8]. Additionally, our approach is security pattern centric. All security estimates are based on used and missing security patterns in places where they are needed. Finally, in this paper, we propose a new security pattern against an attack that we discovered during our experiments and that existing security patterns do not protect against. The rest of this paper is organized as follows: Section 2 describes the systems that we used to experimentally determine the resistance of several security patterns to known categories of attacks. Section 3 contains preliminaries on the fuzzy-set theory and calculations on fuzzy fault trees. In Section 4, the methodology for constructing fuzzy fault trees from UML-class diagrams is described. In Section 5, experimental results are presented, concerning the resistance of security patterns to known attacks, risk evaluation of a no secure and a secure system, and the risk evolution when patterns are introduced in different orders. In the Section, we propose and evaluate a new security pattern named "Secure GET Parameters." Finally, in the Section, we draw some final conclusions and propose future work. In order to experimentally examine the robustness of various security patterns to known attacks, we have developed two systems. The first system, hereafter denoted as no secure application, is a typical e-commerce application with no usage of security patterns, except for Protected System, where various sources for attacks were deliberately included.. If no Secure Pipe pattern is present in the system, a factor to the fault trees for Spoofing Identity, Information Disclosure, and Elevation of Privilege is added, since information could be eavesdropped. Resistance of the Security Patterns Examined against STRIDE Attacks guard to dictionary attacks [7]. The authentication mechanism of a guard can still be marked as of high security. All authentication patterns and, consequently, the Protected System and the Secure Proxy pattern should be resistant to eavesdropping attacks to serve their purpose. Thus, they should always be used together with the Secure Pipe pattern that enforces the use of the SSL protocol. The Secure Pipe pattern offers protection from Information Disclosure attacks. Finally, the Secure Logger pattern offers a strong protection mechanism from reading/tampering the logs, preventing from Tampering-with-Data, Repudiation, and Information Disclosure attacks. Based on the above analysis, we can make conclusions about the resistance of the security patterns under consideration to known categories of attacks. The results are summarized in Table 3. Irrelevant entries to the specific security pattern are left blank. Since we have not considered security patterns that can confront Denial-of-Service attacks, the corresponding category has been eliminated from our analysis. Next, we perform a likelihood-exposure-consequences investigation for attacks that occur in cases where specific security patterns are missing and cases where the security patterns used do not offer total protection. Our investigation is based on the previous analysis, together with knowledge on possible attacks on Web Applications. We note that the likelihood and the exposure (ease) of an attack are the same, regardless of the application, whereas the consequences depend on the data affected and, thus, on the specific application. Although in our investigation, consequences for the specific applications could be considered, we examined the worst case scenario for the consequences, considering that all system data is of crucial importance. Regarding the authentication mechanism, the categories of attacks affected when the authentication mechanism is broken are Spoofing[7], Information Disclosure, and Elevation of Privilege (if someone gets administrator rights). The most trivial case is when no authentication is used at an application entry point. In this case, the likelihood of an attack is very high, the ease of performing an attack is very high, and the consequences are damaging (very high). When the Protected System pattern is used, the likelihood of successfully attacking a guard of this pattern is low, the ease (exposure) of a dictionary attack can be regarded high, and the consequences are very high. When the Secure Proxy pattern is used, two guards must be compromised for an attack to succeed. The likelihood and exposure of compromising the first guard are the same as in the case of a guard of Protected System. The consequences of

attacking the first guard are very low, since the first guard only acts as a front end to the second guard, and no resources are compromised yet when the first guard is compromised. The likelihood, exposure, and consequences of attacking the second guard are the same as in the case of a guard of Protected System. The consequences of attacking the second guard of Secure Proxy are very high, because if the second guard is compromised, then all the protected resources are compromised. In case the Secure Logger pattern is not used in a place where logging is performed [6], the categories of attacks affected are Tampering with Data, Repudiation, and Information Disclosure [8]. If the server where the logs reside is compromised, the log data can be read and changed, letting a user deny having performed an action. The likelihood of such an attack and the ease of such an attack are low, since generally, it is not easy to compromise the server where the logs reside. The consequences regarding Tampering with Data and Information Disclosure are low, since the data kept in the logs is not usually of high importance [10]. The importance of the logs is, however, very high when considering Repudiation (someone could deny having performed an action that he/she performed, or conversely, someone could accuse someone else of having performed an action that he/she did not), and therefore, the consequences are also very high. When the Secure Pipe pattern is not used, the application may not be configured to work with an SSL connection. In this case, important data could be eavesdropped, leading to an Information Disclosure attack, and additionally, if the credentials are eavesdropped, this would lead to Spoofing and Elevation of Privilege[11]. The likelihood of an eavesdropping attack in this case can be considered high, the ease of such an attack is high, and the consequences for all categories affected are very high. When no intercepting validator is used in a path from a class where data is input to a class where this data is shown or a resource (for example, a database) is accessed, having this data as a parameter, then an SQL Injection and/or an XSS attack could occur.

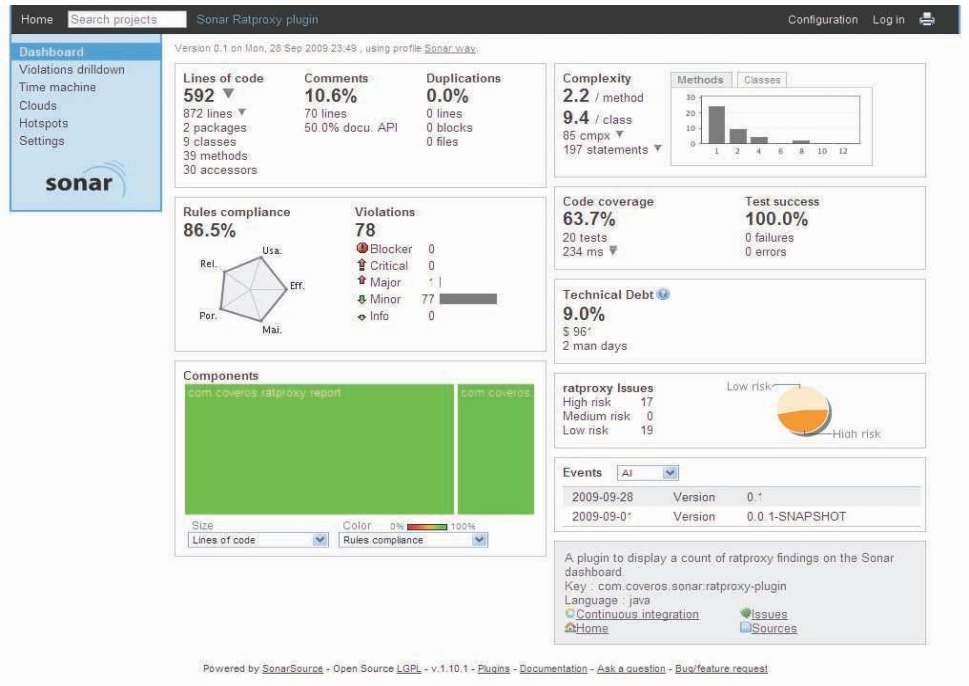

Fig. 1. *The Sonar Quality Dashboard for SecureCI. It displays integrated software vulnerability information.*

Automated CI is often performed during Code check-ins—code checked into a source code control system can be automatically integrated and unit tested to assure its quality. CI done during code check-in typically doesn't test the application's entire feature set but quickly confirms that code enhancements compile and pass a set of unit tests [9]. Nightly builds—each night, software is automatically compiled and a full battery of regression tests are run to ensure the entire code base integrates and operates properly.

2. Methodology

Nightly builds also often automatically execute code analysis to ensure quality and compliance. Weekly builds—for tests that take too long to execute on a nightly basis, weekly builds are often established to compile and test software more fully to manage an automated CI process, CI servers have emerged. Methodology: Driven by these ideas and motivations, we propose a security characterization framework in this article. The framework addresses how to characterize the security properties of components, how to analyze at runtime the internal security properties of a system comprising several atomic components, how to characterize the entire system's security properties, and how to make these characterized properties available at runtime. To inspire trust in a particular composite system, a component's security contract with all the other components, the security provisions that each component requires from ensures to the

others, and the ultimate global security profile of the entire federated system should be clear. Security properties and behaviors of a software system are categorized into 11 classes in ISO/IEC-15408 Common Criteria. These classes are made of members, called families, based on a set of security requirements. We will only discuss a subset of one such security class, user data protection, just to give a snapshot of our characterization framework. The publishable security properties related to user data protection of any atomic component can be categorized as required—a precondition that other interested parties must satisfy during development to access the ensured security services—or ensured—a post condition that guarantees the security services once the precondition is met. Security properties are typically derived from security functions—the implementation of security policies. And the security policies are defined to withstand security threats and risks. A simple security function consists of one or more principals (a principal can be a human, a component, or another application system, whoever uses the component), a resource such as data, security attributes such as keys or passwords, and security operations such as encryption. Based on these, three main elements characterize an ensured or required security property: security operations executed by the components to enforce security properties, security attributes required to perform the operation, and application data manipulated in a compositional contract. Using these elements, we can formulate a simple structure to characterize the security requirements and assurances of individual components'(Oi, Kj, Dk)where *f* represents a security objective formed with three associated arguments; O is the security-related operation performed by the principal i in a compositional contract; K is a set of security attributes used by the principal; subscript j contains additional information about K such as key type, the key's owner, and so on; D is an arbitrary set of data or information that is affected by the operation O; and the subscript k contains additional information regarding D such as whether a digital signature is used or not. The following examples represent a required security property R (protect_in_data) and an ensured security property E (protect_out_data) of a component P: In this example, component P's required property RP states that the data is to be encrypted by any component Q with component P's public key. A plus sign (+) after P denotes public key. The ensured property EP states that component P encrypts the data file with the public key of any component Q. The data is also digitally signed by P with its private key, denoted by the minus sign (−) after P. This format is specific to a particular type of security function related to user data protection. This notation, or a similar one, can be standardized for all components. However, alternative structure might need to be formulated to represent other security classes such as authentication, security audit, trusted path, privacy, and so on.A component that broadcasts an event to receive a service is called a focal component. Software components that respond to the event are usually called candidate components, and they might reside at different remote locations [9]. With the security characterization structure of atomic components previously explained, a CSC between two components such as x and y can be modeled as existing CSC can be referred to as Cx,y.Ry or Cx,y.Ex respectively. The degree of conformity between the required security properties of one component and the ensured security properties of another is the ultimate CSC of the composite system. As is the case of atomic components, we also need to establish a global security characterization of a composite system, because it might be used in further composition as a component. In fact, developers often view this kind of system as a single entity or an atomic component, not as a collection of components in such further components. Current frameworks for software component models such as EJB, Corba, COM, and .Net are limited to the specification and matching of structural interface definitions. Interface description languages (IDLs) deal with the syntactic structure of the interface such as attributes, operations, and events. In our approach, an active interface not only contains the operations and attributes to serve a function but also embodies the security properties associated with a particular operation or functionality. An active interface supports a three-phase automatic negotiation model for component composition: A component publishes its security properties attached with functionality to the external world. The component negotiates for a possible CSC at runtime with other interested candidate components. If it succeeds, the negotiation results are used to configure and reconfigure the composition dynamically. An active interface consists of a component identity, a static interface signature, a static (read-only) security knowledge base of the component, and a (read–write) CSC base that is dynamic based on the information available from the security knowledge base. Before a component is available for use, a certifying authority must certify it. A certificate ensures that the implementation matches the published functionality and the exposed security properties. It is argued that software components can only be tested and certified individually—not within the context of the complete composite system. The certified assurances must be verifiable statically and dynamically. Figure 1 illustrates a skeleton of an active interface structure. The Component in the active interface includes a unique identity (UID) provided by a certifying authority, the component's current residing address (URL), details about the component developer, and the certification authority that certified the component: Component ID (uid, URL, developer_ID, certificate) A certifying authority will verify, certify, and digitally stamp all of this data. It can further reveal more identity information if queried about the certificate, certification stamp, validity period, and so on. All identity and certification information is read-only and public—only the certifying authority can alter it. An interface signature consists of operations and attributes for a particular functionality. These operations and attributes are used for structural plug-and-play matching. These properties are static— read-only properties. Components cannot make any modification to this. This interface is intended to make a structural match before two components are composed. A

security knowledge base stores and makes available the security properties of a component in terms of *f*(Oi, Kj, Dk). The required and ensured properties stored in this KB are specific to the functionality that the component offers. These properties must be based on the actual security functions that the component uses to accomplish a particular functionality. A component might offer various functions, so the exposed security properties can vary accordingly. Once the information is stored in a KB and certified, no other entities can alter its content. Any recompilation of the certified component would automatically erase all certification and identity information stored in Component [8]. If the component needs to alter its security properties, it requires a new certificate after the recompilation. A binary executable piece of code residing in the active interface of the focal component generates CSC conformity results between the focal component and a candidate component. If the system identifies nonconformance between the required and ensured properties it concludes with a security mismatch. The resulting CSC is automatically stored in the CSC base of the focal component, and remains there as long as the composition is valid. Also, a component can accept a partially or completely mismatched CSC, although this might have negative security effects on the global system. If a component becomes obsolete or is no longer needed in a dynamic composition, the associated obsolete CSC might be stored in a log belonging to the focal component for future audit purposes, but it would not be available to any of the participating components. We use a fictitious distributed-system topology as an example of how our proposed active interface would work in a distributed environment. Consider an e-health care system that regards all clinical information passing among the stakeholders, such as the general practitioners, specialists, patients, and pharmacists, as confidential. Assume a focal component Y running on a machine at a GP's office connects with a trusted candidate component S chosen from among many such systems running at various specialists' offices. Y provides a patient's diagnosis report to S to get a prescription. After receiving the prescription from S, Y sends it electronically to a candidate component P residing on a pharmacist's system for a price quotation. Developers would independently develop many such Ps and Ss and make them available from their various distributed sources, potentially able to deliver the functionality that Y wants. However, component Y not only is interested in specific functionality but also wants to know upfront the security properties that those components provide. Assume [3]. In return, Y requires that P digitally sign and encrypt the price data. Note that these security properties of Y are quite different from those for the specialist prescription. Now assume that in response to Y's broadcasting a request for a price quotation, remote components P1 and P3 have registered their interests in providing the functionality that wants. P1 and P3 are developed and serviced by two different development organizations and have their own security requirements and assurances [10]. Y now runs a security test with P1 to verify whether the component could deliver the functionality as well as the security that Y requires. It also verifies whether Y by itself could The entire system scenario is shown in Figure. There are two CSCs in this system: one between Y and S2 (shown by the red dotted line) and the other between P3 and Y (shown by the larger blue dotted line).In the latter composition, S2 is transitively composed with P3 because P3's security requirements partly depend on S2's security assurances, although P3 does not have any direct composition with S2.With the previous examples, we have demonstrated that software components can know and reason about the actual security requirements and assurances of others before an actual composition takes place. The example also suggests that a security characterization is a mechanism to provide "informed consent."2 An informed consent gives the participating entities explicit opportunity to consent or decline to use components after assessing the candidate components' security properties.[A component can accept a partially or completely mismatched CSC, although this might have negative security effects on the global system. Our framework's main objective is to generate computational reflection to let components and their developers identify and capture the various security properties of the other components with which they cooperate [4]. In such a setting, components not only read the met description of others' security properties but also identify security mismatches between two components and evaluate compos ability realistically. Security characterization and third-party certification of components would mutually benefit each other: first, a security characterization would contribute significantly to the process of component security certification; second, certification would make the exposed security properties more creditable to software engineers. When required and ensured security properties are spelled out in simple, comprehensible terms, software engineers are better positioned to evaluate the strength of the security a component provides. They are also well informed about what to expect from and provide to the component to establish a viable composition. In a software engineering context, we must balance security against the other design goals of the entire component-based system. To achieve this, application developers must know about components' security properties. A trusting profile could be gradually built and inspired on the basis of the participating components' self-disclosure of their security properties. The security properties built into a component represent the efforts already put into place to withstand certain security threats. However, the real protection with the committed effort of the component from any security threat is beyond the control of the component. Whether the available resources disclosed by the component are sufficient to withstand a threat is outside the parameters of our framework. A trust-generating effort could only be viable by exposing actual certified security properties of interested parties in a composition as opposed to "secure or insecure" claims. We acknowledge that software engineers' trust in unfamiliar components is understandably difficult to cultivate and that complete trust is undoubtedly desirable, but we believe that our approach would at least contribute to

such trust. One of the real challenges facing the emerging field of software security is the lack of an easily accessible common body of knowledge. Simply put, most software developers and architects—the very people who need to understand and practice software security—remain blithely unaware of their critical role. Without their direct participation, software security will languish. In this installment of Building Security In, we describe a software security portal that the US Department of Homeland Security (DHS) National Cyber Security Division (NCSD) is developing (along with the Carnegie Mellon Software Engineering Institute [SEI] and Digital). The launch of this portal is scheduled for October 2005 as part of the US-CERT Web site. The portal aims to provide a common, accessible, well-organized set of information for practitioners wishing to do software security. In this section, we summarize some of the limitations of the proposed methodology and suggest some extensions and improvements. Our methodology relies on the accuracy of function cycle count measurements. This is possible only if a sophisticated, cycle-accurate simulator is available for the system under consideration, which reports cycle counts for each function excluding the cycles spent by the processor in its descendants. Point your web browser to www.cucwings.com alternatively we can go to www.cucwings.com and click HR in Top Menu Bar. Initially basic data needs to be set up before getting benefitted and utilizing all the options in HR module. To go to Employee Basic Data Set-up page click on the HR Basic Data Set-up link at the Left Side Menu Bar in HR page.

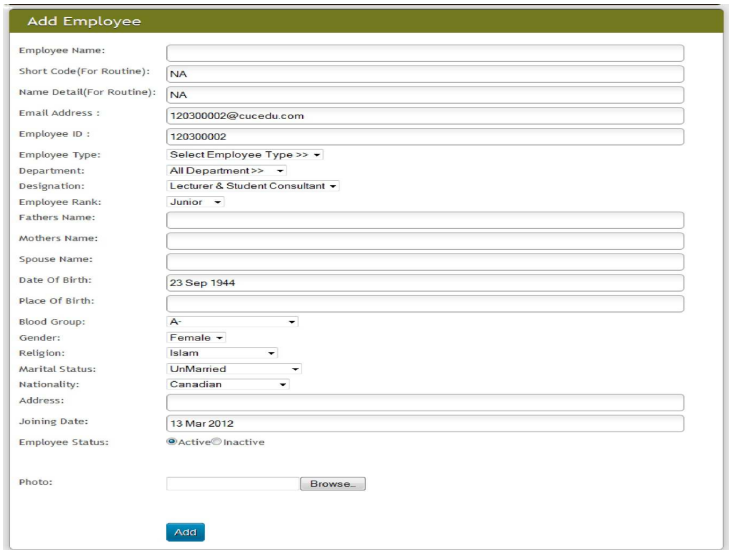

Fig. 2. *Needs of basic data to be set up before getting benefitted and utilizing all the options in HR module*

To add an employee you need to click on the Add Employee button in the left side menu bar of Employee page and you will land in Add Employee page as shown above.

There are different ways to enter information in the system through different fields.

3. Entering Information

Fields	Entering Information and Examples
Text Box	Enter information directly into the field
Drop-Down List	Click ▼ and then select the value from the list
Radio Button	Select one of the values
Check Box	Select to activate/deactivate the option

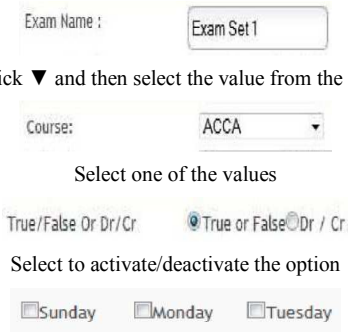

Fig. 3. *Different ways to to enter information in the system through different fields.*

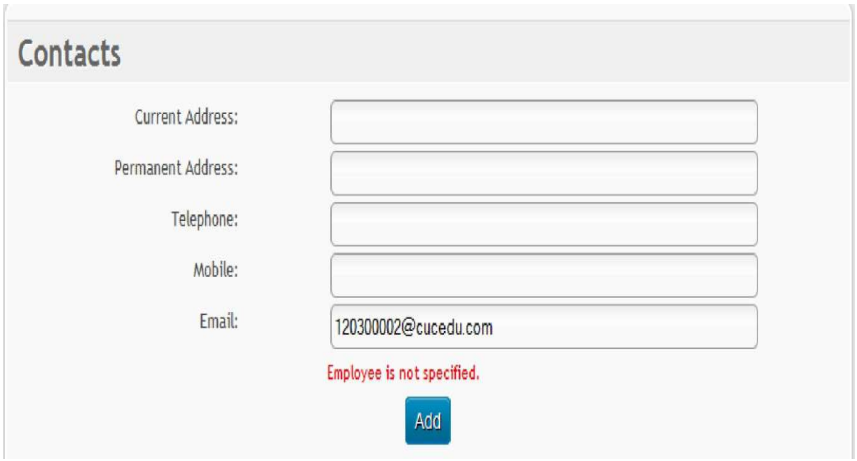

Fig. 4. *Continuous Assessment of security hardening of the ERP software system.*

The system also comes with a 'what you see is what you get editor', which allows user to easily enter and preview larger amount of information.Software-based protection systems are coming into common use, driven by their inherent advantages in both performance and portability. Software fault isolation, proof-carrying code, or language-based mechanisms can be used to guarantee memory-safety. Secure system services cannot be built without these mechanisms, but may require additional system support to work properly. We have described three designs which support interposition of security checks between entrusted code and important system resources. Each design has been implemented in Java and both extended stack introspection and name space management have been integrated in commercial Web browsers. All three designs have their strengths and weaknesses. For example, capability systems are implemented very naturally in Java. However, they are only suitable for applications where programs are not expecting to use the standard Java APIs, because capabilities require a stylistic departure in API design. Name space management offers good compatibility with existing Java applets but Java's libraries and newer Java mechanisms such as the reflection API may limit its use. Extended stack introspection also offers good compatibility with existing Java applets and has reasonable security properties, but its complexity is troubling and it relies on several artifacts of Sun's Java Virtual Machine implementation. Understanding how to create such a hybrid system is a main area for future research. Training throughout the company focused on architectural reviews, secure coding, and testing processes. The training materials were initially licensed from a major university, and have since been customized to their needs. H further customizes the training for product groups, to maximize relevance to the staff. While training is usually a one-time event, organizational turnover is high enough that the training is repeated in each location on a regular basis. In some cases, threat modeling as part of the design process. A company-wide license to use a source code analysis tool, along with training by the evangelist team on how to use the tool effectively[11]. An in-house penetration testing team, coupled with third-party penetration testing when the need arises (e.g., because the in-house team is unavailable).Use of a third-party team to assess the security status of products being considered for OEM or acquisition, to minimize the risk of acquiring security vulnerabilities along with products. This review team currently operates after the OEM arrangement or acquisition has been completed. The evangelist team believes it would be more effective before the deal is signed, but that change has not occurred. Software testing is one of the most fundamental assurances for the high quality of a developed product [2]. Quality of software represents consumer satisfaction across the breadth of a products´ features, including assurances about safety, privacy and security. The commercial software industry typically employs Quality Assurance (QA) technicians through a dedicated QA department. The area of formal testing is identified as a major difference between the commercial and open source projects. The section is by no means arguing against system wide tests but is pointing out the interesting side effects that could result from abusing the system on the commercial side and the extra diligence for the lack of it on the free side. We believe that if QA abuse is true on the commercial side then abiding by good development practices like unit tests and developer diligence while reaping the benefit and the extra assurance of system testing could boost the quality and stress the competitive edge that it has in this area. Consumers can reap the benefits of all of this by having a super reliable system upon delivery that could be deployed with more confidence. Despite the claims by the open source proponents that open source is more secure, a more close examination of the OSS and IP development processes shows advantages and disadvantages on both sides. The claim of open source intrinsic advantage over "closed source" could not be verified from the examined perspectives.

4. Conclusions

Hackers are now targeting the organization's data, putting at great risk of organization and its stakeholders. A secure, formal and structured software development methodology, along with enforceable and pertinent policies was our main target on this project development. A stunning combination

of software assurance is achieved when the above things are combined with a professional certification. In our view "openness", being the most controversial aspect discussed, may not have a big advantage in security. This is evident from the fact that expert "eyeballs" make the difference to the casual developer review. The openness of open source doesn't automatically make it more secure, but it creates an opportunity for motivated individuals to pool together security expertise to do code reviews, security auditing and create tools to help improve security. Two great examples of this are the Sardonyx project On the other hand, disclosing source code can be a slight advantage to the expert hacker in reducing the overhead of analyzing issued patches to produce an exploit for un-patched systems. Lack of formal testing may constitute a disadvantage to open source but produces an implicit advantage by making developers work in a more responsible manner. The numbers come in support of findings that both sides exhibit a mixed set of pros and cons. The record of problems found in OSS and IP don´t suggest the superiority of one over the other when it comes to security [1]. Both open source and IP software have suffered from an abysmal rate of security failures in the last few years. In both worlds the number and sophistication of attacks are on the rise. If software is to meet future needs of business, government and home users, there has to be an order of magnitude improvement in the resilience of software products to attack. Finally we believe that there is a slew of inherent potential on both sides that could be leveraged. There is also room for hybrid models reaping the advantages of both camps. This might be evident from the hybrid development model used with Mozilla Companies like Apple and Sun have taken the initiative to freely publish the source code of projects, indicating potentially closer steps toward a hybrid model. With increased software security incidents, regulatory and compliance requirements, and globalization all changing the landscape of security, one simply cannot take the chance of releasing vulnerable software. Hackers are now targeting your organization's data, putting at great risk your organization and its stakeholders. Damage to your reputation caused by a security breach, and the ensuing loss of customer trust and confidence, might prove irreparable. In today's business environment, software assurance is imperative. In addition to network perimeter security controls, organizations must ensure that software security controls are designed, developed, and deployed to protect their critical information assets. A secure, formal and structured software development methodology, along with enforceable and pertinent policies, must become a part of any organization's operations.

References

[1] Voas, "Certifying Software for High-Assurance Environments," IEEE Software, vol. 16, no. 4, July/Aug.1999, pp. 48–54.

[2] W. Councill, "Third-Party Testing and the Quality of Software Components," IEEE Software, vol. 16, no. 4, July/Aug. 1999, pp. 55–57.

[3] A. Ghosh and G. McGraw, "An Approach for Certify- ing Security in Software Components," Proc. 21st Nat'l Information Systems Security Conf., Nat'l Inst. Stan- dards and Technology, Crystal city, Vir., 1998, pp.82–86.

[4] ISO/IEC-15408 (1999), Common Criteria for Informa- tion Technology Security Evaluation, v2.0, Nat'l Inst. Standards and Technology, Washington, DC, June1999, http://csrc.nist.gov/cc. (current Dec. 2001)

[5] K. Khan, J. Han, and Y. Zheng, "A Framework for an Active Interface to Characterize Compositional Security Contracts of Software Components," Proc. Australian Software Eng. Conf., IEEE CS Press, Los Alamitos, Calif., 2001, pp. 117–126.

[6] C.A. Berry, J. Carnell, M.B. Juric, M.M. Kunnumpurath, N. Nashi, and S. Romanosky, J2EE Design Patterns Applied. Wrox Press, 2002.

[7] Blakley, C. Heath, and Members of the Open Group SecurityForum, Security Design Patterns: Open Group Technical Guide, 2004.

[8] Braga, C. Rubira, and R. Dahab, "Tropyc: A Pattern Language for Cryptographic Software," Proc. Fifth Conf. Pattern Languages of Programming (PLoP), 1998.

[9] P.J. Brooke and R.F. Paige, "Fault Trees for Security SystemDesign and Analysis," Computers and Security, vol. 22, no. 3, pp. 256-264, Apr. 2003.

[10] K.-Y. Cai, Introduction to Fuzzy Reliability. Kluwer AcademicPublishers, 1996.

[11] K.-Y. Cai, "System Failure Engineering and Fuzzy Methodology: An Introductory Overview," Fuzzy Sets and Systems, vol. 83, no. 2,pp. 113-133, Oct. 1996.