Project Report on "Software Engineering for Security: a Roadmap"

SUBMITTED BY:
Chetna Khullar (voa808)
Viswanath Nuggu (tef771)

Instructor: Dr. Gregory B. White

April 29, 2015

PART I

SUMMARY

In this technological world, developing secure software is critical to a company's reputation and business as software applications are the core of any system in today's world. For example, applications like word processors, email servers; web browsers etc are widely used across all domains in business. The impact of a software malfunction or security breach can result in a massive recall, millions in lost revenue, and the loss of sensitive customer data. As a result, applications must be written both in a secure fashion along with security in mind else they may become the weakest link allowing the circumvention of various physical and logical access controls. According to the authors of the paper "Software engineers must be cognizant of these threats and engineer systems with credible defenses, while still delivering value to customers". This paper explains number of research challenges and opportunities that arise due to the interactions between software engineering and security engineering.

The widespread use of internet, cloud computing and smart devices leave all type and scale of businesses more vulnerable than ever before to attacks on their information systems. A company's financial security, intellectual property and level of trust are at risk. Everything can be lost as the result of a successful attack. Sometimes the reason for vulnerabilities in software may be not due to coding problems but with design. So software designers should consider both users and adversaries while designing an application.

In building complex software applications in which more than one vendor is involved in order to meet time to market which resulted in the increased use of commercial, of-the-shelf (COTS) elements where COTS vendors provide only binaries to protect their intellectual property. So the authors of this paper state "Software developers are thus faced with the risks of constructing systems out of unknown black box components". Paper is structured in such a way that it explains security requirements and challenges in each phase of software development life cycle following water fall model.

REQUIREMENTS AND POLICIES

The requirements phase is a crucial phase to create a support structure upon which an application can be built securely. Applications security requirements can be determined after careful consideration of the business context, user preferences, and/or defense posture. The requirements may include the protection of sensitive, private or classified data. If there are multiple clients, the requirements may actually fall under various guidelines and standards so it is important to fully understand the various requirements.

In this phase organization must identify its security policy based on the security requirements of the application to be developed. The TCSEC [1] Glossary defines security policy as "the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information". A security requirement is a manifestation of a high-level organizational policy into the detailed requirements of a specific system. By properly scoping and examining the possible security requirements that an application will need to adhere to, proper architecture and design decisions can be made which help to avert serious design flaws that could otherwise doom a program.

Security Requirements are a kind of non functional requirements. Though some security concerns are addressed during the requirements engineering stage, most security requirements come to light only after functional requirements have been completed. As a result, security policies are added as an afterthought to the standard (functional) requirements.

According to [2], erroneous trust assumptions of developers regarding the manner in which the system will be used usually result in compromise of security. Though developers make such assumptions throughout the software development life cycle, the two main reasons for dangerous and erroneous trust assumptions are: incomplete requirements and miscommunication between developers.

Hence, it is necessary to elicit accurate trust assumptions during the requirements analysis phase.

There are three Security Models and Policies that are commonly used. These include Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Multilevel Security model.

In MAC there are subjects and objects. Each object is associated with security classification such as critical, secret, and public. Subjects can only access when there is an appropriate classification. On the other hand, in DAC access restrictions are based on the identity of the user and subjects with certain access permission who can pass that permission to another subject. In multilevel security model there are objects which are considered readable or writable. Under this security model, each subject and object are assigned a security level. Subjects can only read objects at levels below objects and write to objects at levels above them. If multilevel security model policy is implemented correctly, it would be impossible for information at a higher level of security classification to leak down to a lower channel. The selection of an appropriate security policy and model is very important in early stage of the product's lifecycle. The research paper mentions two challenges in requirements and security policies includes unifying security with systems engineering, unifying security and system models.

Challenge: Unifying security with systems engineering

It is a challenge for systems engineering to develop a secure product by optimizing project resources such as personnel, money and time. Due to this reason, they focus on functional requirements and do not give enough time and analysis for security requirements initially. This paper strongly mentions that the system engineering must be unified with security engineering because security is as important as functional requirements but it is difficult to unify both.

Unifying security with systems engineering will allow the project team to think about security requirements from the initial stages of software planning so that they can allocate timelines accordingly.

Challenge: Unifying security and system models

According to the authors, "the primary challenge here is to extend the syntax and semantics of standards such as UML to address security concerns". Software engineers use system models in the early stages of the development life cycle which include requirement, design, implementation and release in order to improve the quality of the software. They mostly use high-level, object oriented models such as UML. Designing a proper model early in the software development will help us in identifying flaws. Modeling is

also good for reverse engineering. On the other hand, security modeling and policy are added to the system after system modeling finished. We can say that the software engineers are focusing on system models and they always forget about the security modeling at the beginning. If they try to include both the system modeling and security modeling at the beginning, they can build better software. For instance, we can adopt and extend standard UML to include modeling of security related features such as integrity, privacy and access control. This approach will provide benefits such as unified system and security policy design, modularity compactness and reuse in policy representation and leverage of existing standards based tools.

ARCHITECTURE AND DESIGN OF SECURE SYSTEMS

Software architectures and its design can play a vital role in the development of secure systems. Security requirements strongly influence the architectural design of complex IT systems in a similar way as other non-functional requirements. Adding non functional requirements such as performance and reliability into software architectures after the fact is difficult. So, software designers recognized this and incorporated them into software design process long back but this was not the approach with security requirements. They are more often an afterthought. This may lead to serious design challenges for the enforcement mechanism and the rest of the system. The best resolution to this problem is to refine requirements and design processes to bring an earlier focus on security issues. The authors of this paper provided challenges "Legacy Security Mismatches" and "separating the security".

Challenge: Legacy Security Mismatches

Developing uniform policies and enforcement mechanisms for a group of services that span different platforms is the research challenge that the authors identified due to architectural differences between systems. The authors explain the security issues that arise due to the mismatch of architectures between legacy systems and target systems with an example by considering UNIX systems and CORBA architectures. CORBA and UNIX systems have different security policies. While CORBA uses Kerberos based security policy, UNIX uses username, password combination for authorization. There are also other differences between CORBA and UNIX system.

These differences enormously complicate the system that the principals can authenticate themselves with either CORBA or UNIX mechanism. We can create a new system in order to combine two different systems such as user login in one system and also can use other system as well, but building such system which can be used for many platforms is still a research challenge.

Challenge: Separating the Security "Aspect"

The challenges here include modifying the security aspects of a legacy system for security and identifying code relevant to security. Modifying the existing security aspects and integrating the changes back into the system is very difficult. The paper has suggested two new approaches which include aspect-oriented programming and working on architectural connectors to deal with it.

Aspect-oriented programming (AOP) is a programming paradigm that aims to increase modularity by allowing the separation of crosscutting concerns. The AOP forms a basis for aspect-oriented software

development [3]. The core idea of the aspect-oriented programming is to simplify the software evolution. Changing the way a system is distributed, for instance, would involve the difficult task of identifying and changing scattered code concerned with location, connection and distribution. Furthermore, AOP provides the re-scattering location, connection, and distribution back into the code in an automated manner and this strategy is known as aspect weavers.

On the other hand, the other line of research arises in software architecture which include components that form the center of computation in the system, and connector which are the loci of interaction between components. Important security concerns such as authentication and access control arise out of interactions between components. Thus, the security concerns are naturally placed with architectural connectors. The aspect weavers could take on the task of integrating the implementation of these aspects with the rest of the system.

SOFTWARE PIRACY & PROTECTION

Software piracy is a huge challenge to software companies from many decades. Software piracy occurs when people copy, sell, share, or distribute software illegally. It can vary from a limited case of installation of a single-user license on multiple computers to a more chronic problem of widespread online distribution. Regardless of the rationale or delivery method, it is still software piracy.

Although most computer users today are aware that the unauthorized use and duplication of software is illegal yet many show a general disregard for the importance of treating software as valuable intellectual property. Over half of the world's personal computer users, about 57 percent admit to pirating software. Thirty-one percent say they do it "all of the time", "most of the time," or "occasionally" and another 26% admit having stolen software, but only "rarely" [4].

According to the BSA (Business Software Alliance) and IDC (International Data Corporation) 6th Annual Global Software Piracy Study, the retail value of unlicensed software — representing revenue "losses" to software companies — broke the $50 billion level for the first time in 2008. The worldwide losses grew by 11 percent to $53 billion. Excluding the effect of exchange rates, losses grew by 5 percent to $50.2 billion. By observing these statistics, we can understand the complexity of this challenge.

Most people and organizations have a strong economic incentive to commit piracy. There are a number of technologies exist to combat with piracy, however, none of them are can prevent to commit piracy.

Adversary Economics

$$n * C_b >> C_h + n * C_c + P_{11}(n) * C_{11}(n)$$

C_b Cost of one item

C_h Cost of first hack the copy protection

C_c Cost of each item after hacked

P_{11} Risks of getting caught(prosecution Prob.)

C_{11} Possibly subjective cost of each item(fine)

Fig 1. Compares the cost between original copy of the software and piracy software

We can easily see from the fig.1 cost of the original copy cost a lot when we compared with the cost of the first hack from the copy. There are several different techniques that has been used in order to prevent piracy. These include hardware and software tokens, dynamic decryption of code, watermarking – stealth and resilience and code partitioning.

Hardware and Software Tokens

A security token or hardware token is nothing but a "license" associated with the software or hardware. Security tokens are used to prove one's identity electronically. The token is used in addition to or in place of a password to prove that the customer is who he claims to be. The token acts like an electronic key to access something.

The software and hardware tokens technique is the most common one. Basically, when the product sent to the customer at the first time, they also send the license key. Every time when product starts to run, it checks the key. If the key is wrong, the product exits with a license violation error. In hardware token, it will be special kind of hardware which the software checks for the presence of this token before starting. The goal is to increase the cost of breaking the protection mechanism in order to prevent the commit piracy. But this technique is not exactly preventing the piracy and in some cases the attacker may get the tokens.

Dynamic Decryption of Code

In this technique, the software is stored in encrypted form in some digital media and is only decrypted prior to execution using an independently stored key, even sometimes using several encrypted key. This approach also cannot prevent the piracy exactly because during the execution the attacker can access the memory and can encrypted the key.

Watermarking

Watermarking is to embed a secret "watermark" in the software for each customer. It can be used to embed in any kinds of objects that include sound, picture, text and so on. It will be difficult to find and get code from that watermark because they are stealth and resilience.

The Watermarking is raising the probability of getting caught. This approach seems better than the previous two other approaches. However, this technique also cannot prevent piracy exactly because sometimes anonymity becomes common in e-commerce and this does little to discourage pirates.

Code Partitioning

In this approach, some inventors suggested the most important part of the software is placed in inaccessible memory such as ROM or remote memory location which is protected by remote server administered by a trusted party in order to discourage piracy and protect. The author mentions another approach in which the license-checking part of an application is placed in protected hardware. The code can be found by the attacker and the code invokes the protected license-checking code and patch around it. In order to prevent the same, the portion of the application should be physically protected. There are several approaches discussed in the paper around this point.

The first approach discussed in the paper recommends placing a “proprietary" portion of an application within ROM, leaving the rest in RAM but in this case the addresses and instructions could be yielded by a bus analyzer as they are being retrieved from the ROM which allows it to be readily copied. In order to avoid such an attack, it is necessary to protect the memory storing part of the program and the processor executing these instructions.

Another approach suggests delegating the protected part of the program to a remote server administered by a trusted party [5]. In such an approach, before allowing the caller to proceed is checked for valid licensing but it has performance drawback. In another approach, the private key associated with the smart card at manufacture is used to ship the protected part of the software [15]. The major concern of the code partitioning is to decide which portion of the code should be selected for protection which is still open.

There is another approach that is elegant and secure which includes creation of encrypted versions of functions to compute polynomials [16] which are safe from reverse-engineering. However, there is a drawback that the results computed by the polynomials are also encrypted and they must be sent back by customers to the vendor for decryption which causes performance and privacy concerns, ultimately making it unsuitable for interactive applications.

Challenge: Attacker Cost Models.

It is difficult to judge the effectiveness of each approach because the research is not based on an economic model of the adversary's behavior. This can be exemplified in token-based approach which includes self-checking code and some additional measures to inhibit debugging. This includes onetime reverse engineering cost but there is no explanation for how to quantify the cost for a particular implementation of the token based approach. There are several different approaches to dynamic decryption of code but none of them provide a clear model of adversary cost. The actual human cost of removing watermarks is pending for investigation.

According to the author, though there is no cost model yet the best method to protect software is partitioning. Moreover, the suitable cost model should consider current attacks as well as all possible future attacks.

TRUSTING SOFTWARE COMPONENTS

These days, most of software development is about the integration of commercial off-the-shelf (COTS). The middleware technologies COM and CORBA have led to the rise of a wide range of components, frameworks, libraries, etc. Even though using COST products is very beneficial as it reduces the cost, enhances the quality and lesser time-to-market yet the use of the COST products is risky especially in safety-critical systems.

The evaluation of the process for certain customers like utilities, government enforce the disclosure of certain details which are required for its evaluation and to decide if it is safe or not but sometimes these properties are not congruent with the current vendors who face the risk of loss of intellectual property.

According to the DHS, "software security is a serious risk of using COTS software. If the COTS software contains severe security vulnerabilities it can introduce significant risk into an organization's software supply chain. The risks are compounded when COTS software is integrated or networked with other software products to create a new composite application or a system of systems [6]".

According to Voas, the software components are delivered as black boxes which cannot be decompiled back to the source code. There are two options for the COT would be users. They are black box and grey-box. As per Voas, There are two options in the black box approach. Firstly, test the component to make sure it does not misbehave, and secondly, test the system functionality even when the components do not behave correctly, the system can still function correctly.
The latter black box approach has advantages. In spite of the fact, that the testing is expensive and time consuming the also It would subscribe to the overall quality of the whole system. This advantage is over ruled in case the COTS vendor would have adopted aggressive testing practices. In such cases, the gey box approach comes into picture which comprises of the disclosure of the details of testing by the COTS vendor without compromising with the intellectual property.

The grey-box approach uses two techniques:

- Interactive cryptographic techniques
- Tamper-resistant hardware.

Cryptographic Coverage Verification relies on unbiased coin flip in which the basic blocks are chosen at random and the vendor is expected to make a claim and provide test cases to prove the coverage of the randomly selected blocks. In such cases, it is hard for the vendor to cheat and can provide evidence of its stringent quality control.

In tamper resistant, the hardware includes usage of a tamper resistant hardware device which has an embedded proof checker which is faster and simpler than proof creation. In proof checker, a device is presented with the COTS software and the proof. Thereafter, using the private key, the component and a statement is signed by the smart card implying that the property has been proved. These two techniques increase the reliability of the COTS, but there are some challenges with grey-box.

As per the author of the paper, the gray box approaches address only certain aspects of the grey box verification problem. The approaches have been developed only for block and branch coverage but do not cover data flow based criteria [7]. There is no flexibility to the attempts made by the vendor to advance the coverage by including easily discovered code. Moreover, the cryptographic approaches might also be feasible for some verifying the use of other types of quality control methods, such as model-checking approaches.

VERIFICATION OF SYSTEMS

The U.S. government has built high assurance secure systems. The U.S. has published rainbow series and according to Wikipedia: "The Rainbow Series (sometimes known as the Rainbow Books) is a series of computer security standards and guidelines published by the United States government in the 1980s and 1990s." [8].This series of books specify security feature requirements and assurance requirements for its implementation. The Trusted Computer System Evaluation Criteria was referred to as "The Orange Book." However, this was very costly to be used in general purpose systems. In some cases, U.S. government entities (as well as private firms) would require formal validation of computer technology using this process as part of their procurement criteria.

As a result, the U.S. government has been forced to use COTS software to meet cost, quality and schedule. Many of the standards have influenced, and have been superseded by, the Common Criteria. [8]. The Common Criteria (CC) has been internationally standardized and has been used since many years.

Although the evaluation required will be more aligned with the needs of commercial systems, it is not clear that the state of evaluation technology has adequately advanced to make such evaluation practical for every system and companies. The high quality expectations on secure systems has led investigators to apply rigorous methods to show desirable security properties of computing systems and of cryptographic protocols

There are many problems in formal methods involve significant human labor, and expensive. It is based on specifications rather than implementation. Hence, the confidence in formal verification is subject to fidelity and completeness of specs and their relation to final implementation.

The implementations are very complex in practical languages like C++. The engineering approach is to make use of conservative techniques like model checking [9] and static analysis to find defects. These techniques do not guarantee complete elimination of defects but can be useful to defect testing.

Challenge: Implementation-based verification methods

In practical systems, automated tools are very effective to find the security-related defects in the implementations. Model checkers can be used to identify states where security vulnerabilities might exist[10] or hostile library versions can be created of popular API to simulate attacks on a program and to expose vulnerabilities. However, there are some limitations in implementation-based verification method and it takes long time to verify the product.

SECURE SOFTWARE DEPLOYMENT

One common problem encountered during software installation is that installing an update of one component may cause another, apparently unrelated application to fail. The reason behind it is that the two needed applications require different incompatible versions of the same component. The applications using the component would have been created by different vendors. Therefore, installing one application may cause another application to fail, or during the update the current application may also cause some threats in the system. It will be difficult to fix such kinds of problem for many customers.

Most users of personal computers have suffered the consequences of such problems. In addition to this, the personal computers are asked to reconfigure or update their systems to make the system updated against the newly known vulnerabilities. Such kinds of recommendations are often beyond the abilities of an unskilled user.

The resolution of such problems is very time consuming as it may require the user to sit with the technical support personnel and discuss about the intimate details of their PC which can cause information disclosure which in turn can pose a threat to the individual's privacy. There are chances that the help line person may induce the customer to reveal critical details of the installation of a particular PC which can be exploited to gain access to critical data on the user's PC or his network. The task of maintaining a correct, current configuration of software at a machine has been called post-deployment configuration management (PDCM) [11], the automation of which provide timely distribution of both information about correct configurations and software releases. There are two security issues that must be addressed in PDCM systems include controlled delegation of administration and privacy protection.

Challenge: Controlled Delegation

PDCM is a complex system and it is difficult to evaluate the correctness of the sources. For several experts some sources can seem trusted, however, for other experts the same source cannot be trusted. Hence, there should be some facilities for PDCM systems which can allow and enforce flexible delegation and revocation of administrative privileges.

Challenge: Privacy Protection: When users need to get information from the sources, the PDCM system need to protect user's privacy (notions of privacy vary from user to user). The user should be able to specify what information at his or her site can be accessible to whom and the PDCM should enforce such policies making the best use of the technologies.

SECURE COMPUTATIONS, NOT SECURE COMPUTERS

Generally, all software systems include some kinds of bugs which may result in incorrect results due these defects in the software. Some includes more bugs than others. Because of those bugs software system are error-prone. For security point of view, the user might have some concern about the attacks because of the potential threats, the system can work incorrectly. Hence, it will produce incorrect results. In this case, it is required to make sure whether the result of a computation is correct or not. In order to understand this, there should be some criteria and based on those criteria, some tests should be done and the test results should be evaluated.

There exists a term "test oracle" which is a major problem for the software testers [12]. The assurance that the system functions correctly can be obtained by producing a 'proof of correctness' which implies that the system functions correctly in spite of the fact that the system might be under attack. Hence, the process of checking the proof can be automated using an efficient proof checker.

There are certain approaches to distribute the trust among servers like quorum schemes but they are not performance efficient. Another approach is to use secure data structures in which a processor off loads storage of data onto an un trusted coprocessor and the data returned by the coprocessor is to be verified. At a higher level, application servers provide a useful service to generating "proof carrying answers". An

example Necula & Lee's certifying compiler [9]. There are other examples like third party publication of databases which require the publisher to provide compact proofs of correctness along with answers.

PART II

IN DEPTH DISCUSSION OF THE CHALLENGE: UNIFYING SECURITY WITH SYSTEMS ENGINEERING, INCLUDING CHANGES SINCE 2000.

Systems Engineering is the robust approach to the design, creation and operation of systems where as security is the practice of defending information from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction.

Unifying security with systems engineering has resulted in system security engineering, which aims at discovering users security needs designing and making, with economy and elegance, so that they can safely resist the forces to which they may be subjected. This should be an integral and parallel part of overall systems engineering process.

There has been a rapid development and expansion of network-based applications in the last decade. However, this overwhelming success has an Achilles' heel. As a result, there are threats to the software-controlled systems from the internal and external users of the highly connected computing systems. It is very important to unify security with system engineering. However, it is a big challenge to do this. It is hard to unify security with system engineering. Today, with the world-wide internet connection, all the computers, systems and software are faced with the threats and attackers.

According to a research paper on 'Software Engineering for Security' [18] which has addressed 'security of software systems' as an issue of immense concern. It states that it has been observed that as per the traditional approach the security is incorporated in a software system after all the functional requirements have been addressed and the author of the paper argues and emphasizes for the need of security concerns to be an integral part of the entire software development life cycle.[18]

Security of a software system is a multi-dimensional concept [18]. The multiple dimensions of security are:

a. Authentication- It is the process of verifying the identity claimed by an entity.
b. Access control- The process of regulation of the kind of access (e.g.- read access, write access, no access) an entity has to the system resources.
c. Audit trail- It is a chronological record of events leading up to a specific security-relevant system state. This record can later help in the examination or reconstruction of the specific security scenario of interest.
d. Confidentiality- This is the property that deals with making certain information unavailable to certain entities. Integrity- The property that information has not been modified since its inception from the source.
e. Availability- This is the property of the system being accessible and usable for an authorized entity.
f. Non-repudiation- This is the property that places confidence regarding an entity's involvement in certain communication. Security can mean different things at different times.

Generally, security implies one or more of the above dimensions of security. For example, security in e-mail communication would involve ensuring confidentiality, non-repudiation and integrity and security in online shopping would involve ensuring authentication, confidentiality, integrity and non repudiation [18]. The reliable protection mechanisms should be applied to engineer these software systems and at the same delivering the expected value of the software to their customers within the budgeted time and cost.

It is challenging to achieve these two different yet interdependent objectives as the current software engineering processes do not provide enough support for the software developers to achieve security goals. The software engineering and security engineering are ever-evolving disciplines and software security engineering is still in its infancy. There are many proposed models for the unification of the process models of software engineering and security engineering which help to improve the steps of the software life cycle that would better address the underlying objectives of both engineering processes. The unification helps to incorporate the advancement of the features of one engineering process into the other [17].

According to Greg Morrisett in a discussion on the same topic, he states:
"The more we rely on software, the more we need mechanisms to ensure that software is trustworthy."

According to the traditional approaches to software security, it has been assumed that users could easily determine when they were installing code if the software is trustworthy. This assumption held some stand when computers controlled few things of real value, when only a small number of people (typically experts) installed software and when the software itself was relatively small and simple. But the security landscape has changed dramatically with advances in technology (e.g., the explosive growth of the internet, the increasing size and complexity of software) and new business practices (e.g., outsourcing and the development of open-source architecture). And, with the advancement of technology, the more we rely on software to control critical systems, from phones and airplanes to banks and armies, the more desperately we need mechanisms to ensure that software is truly trustworthy.

It is very important to unify security with system engineering. However, it is a big challenge to do this. It is hard to unify security with system engineering for the following reasons:

a. The lack of resources, time, funding, personnel and the technology is limited for a company. So, the company always wants to use the limited resources to deliver the most valuable software to the customers.

b. If we want to design a "truly" secure system, it would be very expensive. A company wants to minimize the cost. For them, adding secure features means adding additional cost.

c. Security requirements have not received the same type of careful analysis as did in system engineering.

d. Some people regard security as function disabler. Software Engineering considers security as non-functional requirement. Nonfunctional requirements introduce quality characteristics, but they also represent constraints under which the system must operate.

e. Security is still an afterthought. Even though the importance of security is widely acknowledged, few projects address it with the appropriate priority. So security is still an afterthought.

f. The security mechanism needs to be fitted into pre-existing design; this leads to design challenge and might introduce software vulnerabilities.

g. Security requirements are generally difficult to analyze and model. The developers are not security specialist and lack expertise for secure software development. This increases the difficulty of integrating security into system engineering.

h. Attack Techniques: It is alluring for the malicious hackers because of financial incentives to break into and control personal and business computers. Once they break into a machine, they can gather personal information (e.g., passwords, credit card information, and social security numbers) and use it to clean out bank accounts, apply for credit cards, or gain access to other machines on the same network. In addition, if attackers break into and control machines, they can use them to send "spam" (unsolicited e-mail advertisements), to store illicit digital goods (e.g., pirated music or pornography), or to launch denial-of-service attacks against other computing systems.

i. One technique attackers use to gain a foothold on a machine is to trick the user into thinking a piece of code is trustworthy. For example, an attacker may send an e-mail that contains a malicious program, but with a forged return address from someone the user trusts. Alternatively, an attacker may set up a website with a name that appears to be trustworthy, such as "www.micr0s0ft.com" (note that the letter "o" has been replaced by the numeral "0"), and send an e-mail with a link to the site suggesting that the user download and install a security update. Even sophisticated users can be fooled.

j. Even good guys can't be trusted

k. Another technique that the attackers use to gain control of a machine is to find a bug in an already installed program that communicates with the outside world. An example is a buffer overrun in a service, when programmers fail to put in a check and the input is too long, the extra bytes overwrite whatever data happen to be stored next to the input buffer. The clever attacker will enter an input long enough to overwrite the return address with a new value, thereby controlling which code is executed when the input routine ends. In an ideal attack, the rest of the input contains executable instructions, and the attacker causes control to transfer to this newly injected code. In this way, the attacker can cause the program to execute arbitrary code [18].

The reasons given above show it is a big challenge to unify security with systems engineering. It is believed that security should be considered during the whole development process. And the security

features should be defined together with the requirement specification. If there are some conflicts between security features and functional requirements, we can identify them in advance. If adding security features as an afterthought, it will increase the chances of conflicts and might cost a huge amount of money.

According to a research paper, Software Architecture can help solve several problems that lie in the path of developing secure software [18].

In the paper "Software Engineering for Security: a Roadmap", the author stated: "currently, these compromises are made on an ad-hoc basis, mostly as an afterthought".

There are things that have changed since the year 2000 in terms of the security unification with the system engineering. The giant business tycoons and brands are considering security and giving it a great importance.

Security engineering and software engineering activities should be integrated into the systems engineering life cycle from concept to retirement. The industry has recognized the importance of security in system engineering from past one decade. Some companies are following a structured yet agile approach to planning, requirements definitions, systems architecture & design, systems development and integration, verification and validation, deploying, maintaining, and retiring secure solutions.

The security is taken into consideration and is being inculcated in the software development right from the requirement analysis to the deployment of the software by the software market giants like Apple and Microsoft. The below discussion describes in detail the various products of the companies which have been launched in the market in last few years. The software and the products launched reflect a huge investment of time and money on unifying security with the system engineering.

In this competitive world, if the company launches a product with security vulnerability then that would lead to huge losses in the revenue and reputation. This unification of security and system engineering is only possible if there is a methodology for enumerating the vulnerabilities of a system and the system engineering countermeasures the approach that can best close those vulnerabilities.

Hence, the below discussion describes in detail the feature of the products launched and the software security features incorporated into them which involves data encryption and other methodologies implemented for security of the product.

There has been a message from Tim Cook about Apple's commitment to customer's privacy and security. According to him, the Security and privacy are fundamental to the design of all their hardware, software, and services, including iCloud and new services like Apple Pay. And they continue to make improvements. Two-step verification, with which the Apple encourages all its customers to use, in addition to protecting their Apple ID account information, now also protects all of the data they store and keep up to date with iCloud.

Every Apple product is designed around the principles that the user is told upfront what's going to happen to their personal information and the user is asked for the permission before it is shared with the user. And in case the user does not want to share the information, it has been made easier for the user to disable

sharing of data with the Apple. The sole aim behind asking for customer data is to the user with a better user experience [20].

According to the latest page on Apple's website, they state that they would be publishing the website to explain how they handle our personal information, what do they do with the information and what data is not collected, and why. The Apple is trying to make sure that the customers get the updates on Apple's website about privacy at Apple at least once a year and the customer is would be provided information about the significant changes to their policies [20].

The Apple makes a strong claim that Apple have never worked with any government agency from any country to create a backdoor in any of their products or services and they have also never allowed access to their servers they are assuring that they would never allow it ever. The Apple states, "The moment you begin using an Apple product or service, strong privacy measures are already at work protecting your information. We build extensive safeguards into our apps and the operating systems they run on".

Security has been taken care in all the application of Apple and there are strong claims made by Apple that their applications are safe and secure [21]. The below products from the Apple involve customer's sensitive information and money. The Apple explains the various feature it offers in its products and at the same time how they have unified security in their system engineering in order to support their claim.

<u>iMessage and FaceTime</u>
According to Apple, the communications on iMessage and FaceTime are protected by **end-to-end encryption** across all devices. With iOS 8 and Watch OS the iMessages are also encrypted on the device in such a way that they can't be accessed without your pass code. Apple has no way to decrypt iMessage and FaceTime data when it's in transit between devices. So unlike other companies' messaging services, Apple doesn't scan the communications, and they wouldn't be able to comply with a wiretap order even if they wanted to. While they do back up iMessage and SMS message for the customer's convenience using iCloud Backup, it can be turned off whenever the user wants. And they don't store FaceTime calls on any servers [21].

<u>iCloud</u>
Apple never scans any of customer iCloud data for advertising. All iCloud content is encrypted in transit. In case Apple uses third-party vendors to store customer data, Apple **encrypts** it and never gives the keys to the third-party. Apple retains the encryption keys in their own data centers. In addition to this, it suggests strong, unique passwords for the websites the customers use. In addition to a strong password, two-step verification gives another layer of protection for the iCloud account [21].

<u>Safari</u>
Safari is the first browser ever to block third-party cookies by default and offer **private browsing**. Safari is built to offer the safest browsing possible. It prevents suspicious sites from loading, and uses sandboxing to help keep harmful code confined to a single browser tab so it can't reach the rest of the user's data [21]

<u>Mail</u>

In order to protect the user's privacy, all the traffic between any email application and iCloud mail servers is **encrypted**. The mail servers are updated to support encryption in transit with other email providers that also support it.

Apple Pay

It lets you pay in an easy, secure, and private way. And it works on iPhone 6, iPhone 6 Plus, iPad Air 2, iPad mini 3, and Apple Watch. In this, the actual card numbers are not stored on the device or on Apple servers. Instead, a unique Device Account Number is created, encrypted in such a way that Apple can't decrypt, and stored in the Secure Element of your device. The Device Account Number in the Secure Element is **walled off** from the iOS device and Apple Watch, is never stored on Apple Pay servers, and is never backed up to iCloud.

When the user pay in stores the device account number and a transaction-specific dynamic security code are used for processing the payment. So neither Apple nor the apple device sends the actual credit or debit card numbers to merchants. Apple Pay retains anonymous transaction information such as approximate purchase amount.

The above data and information shows that how the security has become an important part of any software and this security is not only in terms of hardware but it is now getting embedded in every single software and hardware so that the user's data is safe and secure.

Microsoft is working a lot on the security front. They are also keeping the security on their top list. The latest tools and techniques released by the company reflect the inclusion of security in to their development and business models which implies that the security has been included in their software engineering process and they are keen to spend time and money towards it.

Below mentioned are the various products and techniques that have been launched by Microsoft and which shows the security as an important aspect. It reflects the change since the year 2000.

Microsoft Security Essentials (MSE) is an antivirus software (AV) product that provides protection against different malwares like computer viruses, spyware, rootkits and Trojan horses which was initially launched on 29 September 2009

Below mentioned are various tools that are in market by Microsoft which help in security, which were developed after 2000:

- Attack Surface Analyzer 1.0 (Date Published: 8/2/2012) - Understand the attack surface before & after new apps are deployed.
- Microsoft Threat Modeling Tool 2014A (Date published 4/14/2014) – This is the tool that helps the engineers to find and address system security issues.
- MiniFuzz basic file fuzzing tool (Date Published- 8/24/2011) – A simple fuzzer designed ease adoption of fuzz testing.
- Regular expression file fuzzing tool (Date Published- 8/24/2011)- A tool to test for potential denial of service vulnerabilities.

The below figure explains the Scope of Improving Web Application Security: Threats and Countermeasures in Microsoft applications. It covers the Web server, remote application server and database server. At each tier, security is addressed at the network layer, host layer, and application layer.

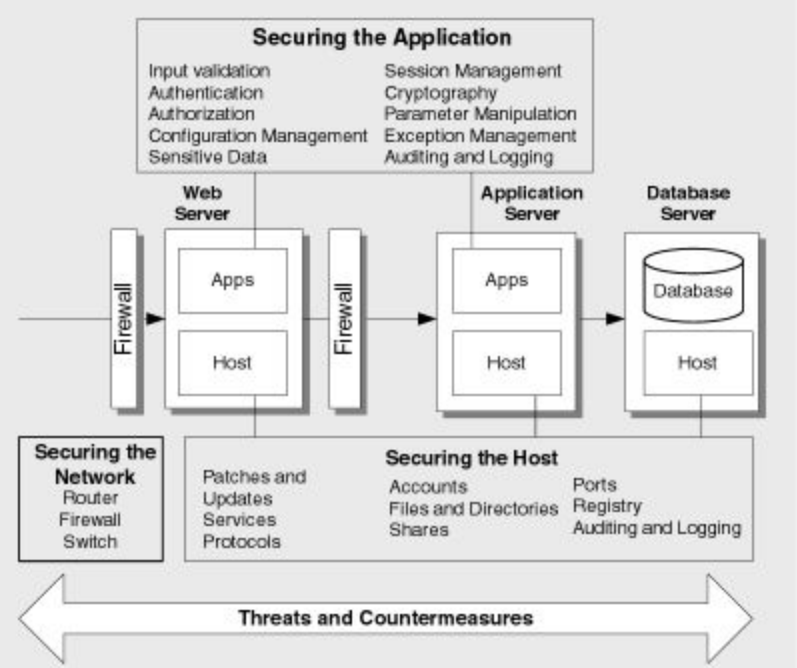

Fig 2. Process to improve security in web applications [24]

As per Wikipedia "The Microsoft Security Development Lifecycle is a software development process used and proposed by Microsoft to reduce software maintenance costs and increase reliability of software concerning software security related bugs" which was released on 2008-04-15 [24].

The above discussion shows clearly the system engineering is being unified with the security to a very large extent nowadays as compared to the year 2000. There are huge improvements and advancements in the field and a lot of money is being pushed into its research whose results would definitely help to improve the unification in the coming years. The below section enumerates the recommendations for the same.

PART III

RECOMMENDATIONS FOR SECURITY PROFESSIONALS REGARDING "UNIFYING SECURITY WITH SYSTEMS ENGINEERING"

Below are the recommendations towards the improvement of the unification of the security with the systems engineering.

Systems engineering is an interdisciplinary field of engineering that focuses on how to design and manage complex engineering projects over their life cycles. We recommend identifying and implementing security requirements in each phase of systems engineering. There are seven phases in systems engineering process which include state the problem, investigate alternatives, model the system, integrate, launch the system, assess performance and re-evaluation.

In the first step of stating the problem we need to identify the customers, understand customer needs, establish the need for change, discover requirements and define system functionalities. We recommend that we should include identification of security needs in this phase and should get the customer to answer a questionnaire to discover the critical security features that he wants.

For the second phase "investigate alternatives", apart from investigate the functional alternatives; we recommend investigating the non-functional alternatives, especially the security alternatives. In addition, a risk assessment should be done in this phase to identify the potential conflicts between the functional requirements and the security requirements.

For the third phase "model the system". In system engineering, this phase is to use models to clarify the requirements, reduce cost and reveal the potential duplication of efforts. However, we recommend to keep in mind is if the security is not kept in mind, the cost would be very high after software ships.

For the phase "assess performance". We recommend that criteria for security performance should be set up. The security features like confidentiality, availability and etc should be adequately assessed. For re-evaluation phase, in system engineering this phase usually focus on continually and iteratively to re-evaluate the functional requirements of the system for future improvement. We recommend that the software developers should not only evaluate the functional feature but also the security features. Since the functional features have been updated, so should the security features. In this case, it is less likely to introduce new vulnerabilities to the existing system.

As another approach we would like to make more recommendations for the improvement for every software engineering phase. As we have first phase to discover needs, we recommend discovering information protection needs as well. For the phase of defining system requirements, the recommendation would be to include system security requirements as well.

Moreover, when the software system architecture is designed, it would be more unified with the security if the system security architecture is aloes designed along with it. In addition to this, for the detailed design development in the system engineering, it would be a good idea to include and design detailed security design. And when the system is being implemented, the systems security should also be

implemented. The last but not the least, when the system is accessed for effectiveness and it should access the information protective effectiveness.

We also recommend including a security expert in the software development team, and letting him guide the development process. We recommend that the whole project team should be trained in security. All the associates from managers to the analyst, designers, developers, testers should be well acquainted with the security knowledge so and they should be trained to integrate security engineering in every phase of system engineering. Moreover, a dedicated review session for identifying security issues should be included with every phase of the software development lifecycle with continual inspection. Including timelines for unifying security into systems engineering is also required. However, there are many challenges to build a secure system. In this report, we mainly discussed the challenge of unifying security with systems engineering. Even there are changes in this area since 2000; there are a lot of researchers that are aiming to solve technical issues in this area. Hope the software developers can overcome this challenge in the future.

CONCLUSION

Unifying the two fields of Software Engineering and Security is a challenging task. In general software developers focus on the functional requirements and forget about the security. As a result of that security, become a serious problem after product complete and after software release. In order to eliminate serious security threats, developers must focus on the security from the beginning as they did for functional requirements. The government must force the software companies to think on security at the beginning. Even though there are some difficulties to combine software engineering and security, if developers give enough effort to implement security at the beginning, it will not be problem anymore.

There are improvements in the unification of the security with the system engineering in last decade and the software companies are inculcating security in their system engineering and this area needs further research to make systems more secure.

REFERENCES:

1. Department of defense trusted computer system evaluation criteria. Dept. of defense standard, Department of Defense, Dec 1985.
2. J. Viega, T. Knono, B. Potter. Trust (and MisTrust) in Secure Applications. Communications of the ACM. February 2001.
3. http://en.wikipedia.org/wiki/Aspect-oriented_programming
4. http://www.safenet-inc.com/software-monetization/software-protection/
5. Monden. A secure keyed program in a network environment. In Proceedings of the Twentieth International
6. Conference on Software Engineering, 1998. http://en.wikipedia.org/wiki/Commercial_off-the-shelf
7. P. Frankl and E. J. Weyuker. An applicable family of data ow testing criteria. IEEE Transactions on
   Software Engineering, August 1988.
8. http://en.wikipedia.org/wiki/Rainbow_Series
9. P. A. Godefroid. Model checking for programming languages using verisoft. In Proceedings, POPL 97,1997.
10. M. B. Dwyer and J. Hatcli_. Slicing software for model construction. In Proceedings of ACM SIGPLANWorkshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'99), 1999
11. R. S. Hall, D. Heimbigner, A. van der Hoek, and A. L. Wolf. Architecture for post-development configuration management in a wide-area network. In 17th International Conference on Distributed Computing Systems, May 1997
12. E. J. Weyuker. On testing non-testable programs.The Computer Journal, 25(4):465{470, 1982.
13. G. C. Necula and P. Lee. The design and implementation of a certifying compiler. In Proceedings of the '98 Conference on Programming Language Design and Implementation, 1998.
14. Pickholz. Software protection method and apparatus. United States Patent 4,593,353, 1986.
15. S. M. Matyas and J. Osias. Code protection using cryptography. United States Patent 4,757,534, 1988.
16. T. Sander and C. F. Tschudin. On software protection via function hiding. In Information Hiding, pages 111{123. Springer-Verlag, 1998
17. http://www.igi-global.com/chapter/software-security-engineering/18390
18. http://www.dsc.ufcg.edu.br/~jacques/cursos/map/recursos/SoftwareEngineeringandSecurity.pdf
19. Bush, W., J. Pincus, and D. Sielaff. 2000. A static analyzer for finding dynamic programming errors. Software—Practice and Experience 30(7): 775–802.
20. https://www.apple.com/privacy/
21. https://www.apple.com/privacy/privacy-built-in/
22. http://www.microsoft.com/en-us/sdl/
23. https://msdn.microsoft.com/en-us/library/ff649874.aspx
24. http://en.wikipedia.org/wiki/Microsoft_Security_Development_Lifecycle
25. http://www.sie.arizona.edu/sysengr/whatis/whatis.html