Securing the Rapid Application Development (RAD) Methodology

BY

Kividi Kikama Jr

Lewis University

December 8, 2010

(This page has been intentionally left blank)

TABLE OF CONTENT

ABSTRACT

Every organization, big or small, depends on some kind of software application as part of its everyday operation. Whether it's the software in the cash register, the gas pump, or the e-commerce website, these applications help the business to increase productivity and respond to the needs of a larger customer base in an expedient and efficient manner. Software development has been around for the last half century in one form or another, and many of us are too young to remember a world where the tedious tasks that we hardly give a second taught to were done manually. The software development process is a critical part of an organization's success because the development of software is usually an attempt to respond to a problem, address a shortcoming or to gain a competitive advantage. The objective of software development is to not only resolve the technical problems, but it also addresses the organizational problems and, in the longer term, the business problems. Developing usable software requires a well-defined process where all the parameters and objectives are clearly defined. Developing applications, specifically web applications, requires careful planning and a structured process in order to obtain desired results in a timely and cost-effective manner.

The Software Development Lifecycle (SDLC) is just such a structured process that is used by most application developers in order to respond to the challenges that they are presented with by the business leadership. There are several models of the SDLC in use by different organizations such as the Waterfall Model, Spiral Model, Top-Down Model, Bottom-Up Model, and Rapid Prototyping which is the subject of this paper.

The evolution of SDLC models has gone hand-in-hand with advancements in technology and continuous research done to address the shortcomings of previous models. Another reason for the continued progress in the models of SDLC is that the business leadership of many organizations, in their attempt to cut costs and streamline operations, is putting a lot of pressure on developers to reduce development time and provide a quicker turnaround on projects. Often times, these projects are required to be completed at a lower cost and without sacrificing quality. In response, agile methodologies such as Rapid Prototyping and Extreme Programming (XP) have increased in popularity among developers.

Rapid Application Development like most agile methodologies presents some unique security concerns and most security methodologies are built for traditional development methodologies, which are qualitatively and quantitatively different from agile development methodologies (Alberto Sillitti, 2010). It is therefore necessary to adapt the RAD methodology in order to account for the shortcomings in the area of security by adding simple but effective security measures in each phase of RAD. This paper will introduce the activities aimed at improving the security of the RAD methodology by baking security activities into the core of the development process.

CHAPTER ONE

INTRODUCTION

Building a secure software application involves incorporating carefully planned activities in the design process. Consideration of security in the System Development Life Cycle is essential to implementing and integrating a comprehensive strategy for managing risk for all information technology assets in an organization (Richard Kissel, 2008). This chapter explains why it is necessary to account for security in the software development process.

1.1 Purpose and Scope

Software development has greatly improved the quality of our lives. It has made the world feel smaller and has increased our connectivity to one another. This connectivity has not only been positive, it has also unearthed some unintended consequences that individuals as well as organizations have to fight against. Software development has eroded the boundaries that once existed due to time, location and distance. Nowadays, transcontinental transactions are performed in a matter of seconds, blurring the dividing lines that once separated the different entities.

Despite the many benefits of Software Development there are side effects that can become a burden if they are not handled correctly. The same software that is used by the organization to increase productivity has become an access point for criminals seeking to penetrate the organization in order to steal or sabotage critical assets. As software has increased in usability and complexity, so too have exploitable weaknesses and vulnerabilities. Studies have shown that the number of software vulnerabilities continue to trend upward at an alarming rate. The number of

vulnerabilities reported to CERT from 1997 through 2006 has shown a steep increase from year to year creating an urgent situation for organizations to implement adequate software security measures in order to mitigate the risks and protect the critical infrastructures that run our businesses (Allen, 2007). Figure 1.1 shows the number of vulnerabilities reported from 1997 to 2006.

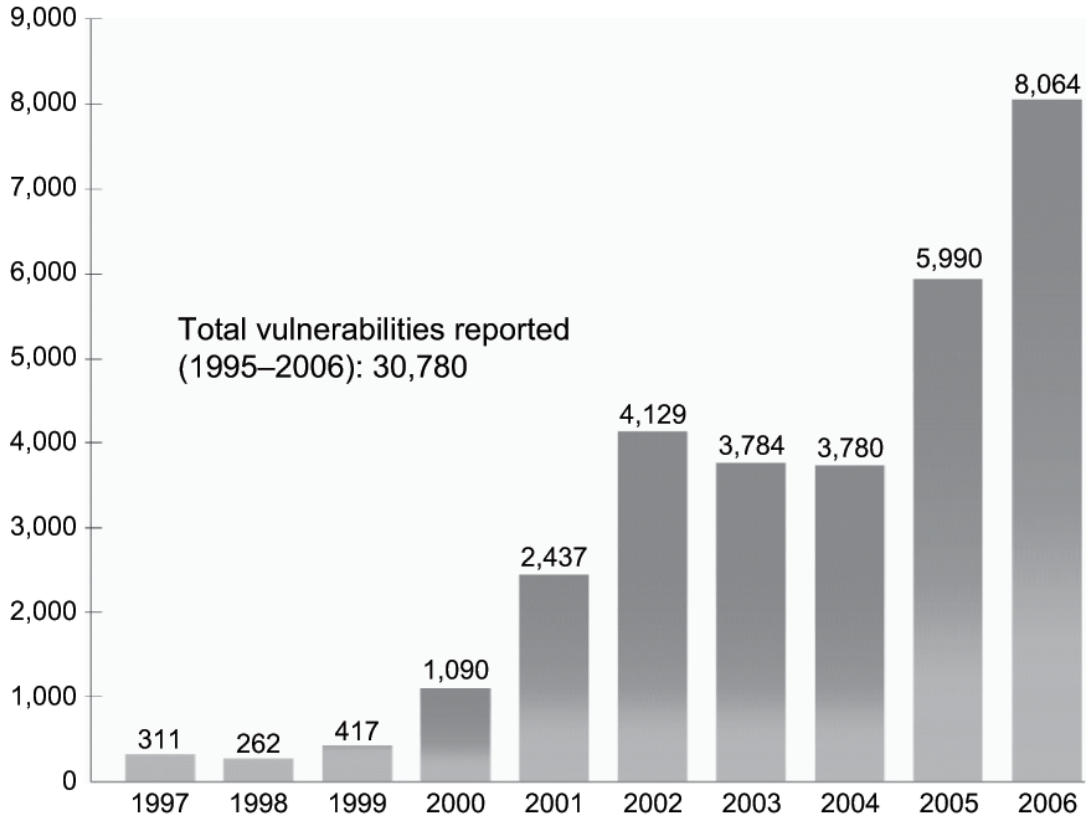

Figure 1.1 Vulnerabilities reported by year (Source: McGraw, 2006)

In the past three months, over 540 million attacks were blocked in 228 countries. Last quarter, even Norfolk Island with a population of 2,141 appeared on Kaspersky Lab's antivirus radar. During the quarter, the average number of infection attempts increased globally by 4.5% per month (Namestnikov, 2010).

In the Global Security Survey report conducted in 2007 by Deloitte, 87 percent of respondents cited poor software development quality as a top threat for the next 12 months (Allen, 2007). This is as true today as it was 3 years ago and creates the need to include security as part of an organization's software development life cycle in order to ensure that these vulnerabilities are handled appropriately.

The purpose of this paper is to serve as a guideline for Software Developers and Project Managers on how to integrate security into the Software Development Lifecycle (SDLC) RAD methodology while taking into account the faster turnaround from conception to market that is a characteristic of this methodology. I will present ways to introduce security measures within each phase of the model which, in turn, will ensure that the end process is secured. I will show how incorporating security activities into the SDLC will increase the security posture of the organizations by ensuring stronger security, reducing the likelihood and/or impact of exploited vulnerabilities. I will also show that the placement of correct security measures in each phase of the RAD methodology is more beneficial when done during the development process than attempting to add it at a later stage. Furthermore, I will present evidence to show that the end result of inserting correct security measures in the SDLC is the reduction in overall cost to the organization.

CHAPTER TWO

SOFTWARE DEVELOPMENT

2.1 History of Software Development

Most people under 50 years old have never known a world without automation. In earlier times, most business processes required manual intervention by a multitude of people and resources, thereby increasing the production time and decreasing the quality of services that businesses where able to offer. Software development as we know it today dates back to the late 1940's and early 1950's and was a result of the need for businesses to streamline their operations and increase productivity and the quality of their products and services . The productivity of software projects has increased about 3 times since 1970. In 1970, COBOL was the state of the art, mainframes were in vogue, and the PC was nothing more than a dream of software engineers. The internet didn't exist. By year 2000, end-user computing exploded. Software developers are developing in languages like Java, C++, ASP, and other visual languages. No longer are software applications being developed for back office operations, but software applications are being used as marketing tools and competitive weapons. Applications are increasingly delivered to customers via the internet (Longstreet, 2006).

From its humble beginnings, software development has gone through tremendous growth over the years, increasing in size, complexity, and functionality, to the point that many have a hard time imagining how things were done before software development became a common practice. The rise of software development to the point of becoming an integral part of the business culture has been greatly influenced by major developments and discoveries in hardware development and the rise of the Internet. These

developments led to an increase in demand for reliable software to run these new machines, and, in response, many software were written usually without any established model. It was a little difficult to develop software without a proper model, so the NATO Science Committee sponsored two major software conferences, one in 1968 and the other in 1969 that many consider as official birth period of software engineering (History of software development, 1997). The attendees at these meetings were senior figures in computing. They discovered a remarkable set of similarities in the problems that they were having trouble dealing with. They thus legitimatized Software Engineering as the study of the broad range of problems encountered in developing software. The 1968 NATO Conference also devoted considerable attention to many issues that are quite familiar to us today. Thus the issue of how to create processes that could be expected to be effective in producing high quality software on schedule and within budget was highlighted (Osterweil, 2007).

The advent of the World Wide Web opened many doors for the field of Software Engineering. The World Wide Web brought out opportunities like never before with many programmers hired to implement web applications that have taken most business to the virtual sphere. In addition, the internet has also opened paths for hackers to attack the infrastructure of many organizations through their public-facing web applications by exploiting vulnerabilities in their software.

2.2 Software Development Life Cycle

Software development (also known as Application Development; Software Design, Designing Software, Software Engineering, Software Application Development, Enterprise Application Development, Platform Development) is the development of a software product in a planned and structured process.

This software could be produced for a variety of purposes. The three most common purposes are to meet specific needs of a specific client/business, to meet a perceived need of some set of potential users (the case with commercial and open source software), or for personal use (e.g. a scientist may write software to automate a mundane task) (Software development, 2010).

The Software Development Life cycle (SDLC) is a structured business process that is used by many organizations that build software. The SDLC has been around since the 1960's and has morphed into several versions over time as many organizations have adapted the process to fit their specific development needs. From the sequential Waterfall model which was the original version to the iterative models such as the Spiral model and on to the agile models such as Extreme Programming there are many models in use today. The goal of a good SDLC process is to capture, verify, and implement all the requirements needed to make the application useful to the organization (Purcell, 2007). The SDLC requires the involvement of people from many disciplines such as architects, analysts, programmers and users working together toward a common goal. Often times, the success of the final product depends on the choice of model, so organizations need to be mindful of selecting the model that fits their situation. Factors such as time, skills, and experience in software development play a part in determining which model will work for software development.

2.3 Stages of the SDLC

The Standard SDLC consists of five stages as shown in figure 2.1. These stages can be customized according to the specific model.

Figure 2.1 The SDLC – A Conceptual View (Source: Richard Kissel, 2008)

II. The Five SDLC Phases (Richard Kissel, 2008)

The five phases of the SDLC, as defined by NIST SP 800-64, are as follows:

1. Initiation

The main objective of this phase is to identify all of the requirements needed to design or purchase the system. This is accomplished by first determining the reason for the system, identifying the business problem that the project is attempting to resolve. The second activity covered in this phase is to identify all the stakeholders that are affected by the undertaking. Steps in this phase include establishing the basic system idea, preliminary requirements definition, feasibility assessment, technology assessment, and management signoff to continue to the next phases (Purcell, 2007).

2. Acquisition/Development

The main objective of this phase is to convert the functional and technical requirements from the initiation phase into detailed plans for the proposed system that can be interpreted by software programmers. Steps in this phase include analyzing the results from interviews, developing mock ups

and use cases, and translating the results into sequence diagrams, activity diagrams, state diagrams. Another activity that's prevalent is this stage is the refinement of the user interface design to include more details. At the end of this phase the development team should decide on the direction for implementing the best solution that responds to the business problem outlined in the initiation phase.

3. Implementation

The main objective of this phase is to create a working application using the analysis and design recommendations from the previous steps. Steps in this phase include the actual coding of the information system by programmers, preliminary testing and debugging to ensure that everything is working as expected. User functionality is tested through user acceptance testing, Quality Assurance testing, load testing, and other types of technical testing. The end result of this phase is the integration of the completed system into the production environment.

4. Operations/Maintenance

In this stage the system is live in the production environment; the main objective now is to make sure that it continues to function as planned. In order to ensure that the system remains functional preventative and maintenance steps must be performed in a structured way. Steps in this phase include implementing patches, and correcting bugs that are discovered along the way. It does not include functionality upgrades or additions; those must follow the normal development phases starting from initiation. As long as the system exists in a production environment it must be maintained. The end result of a proper implementation of this phase is a dependable system that runs with minimal interruptions.

5. Disposition

The main objective of this phase is to remove the system from the production environment in a structured way. This occurs once the system's functionalities are longer needed or a replacement system has been created. Steps in this phase include archiving the existing system, and performing a switch to a new system by minimizing downtime. The end result of this phase is the complete removal and retirement of the system.

2.4 SDLC Models

There are many ways of implementing the Software Development Life Cycle (SDLC) in an organization. Each organization has its own set of realities so a standard one fits all SDLC is not realistic. Often times in order to respond to their specific needs organizations implement their own version of SDLC. The Software Development Life Cycle can be carried out in a number of different ways and these ways are called models. The SDLC models have evolved over the years as technological advances have shown the limitation and weaknesses of older models. From the linear-sequential models such as the Waterfall, through the iterative models, newer agile models have drawn from and improved on the best aspects of older models creating models that provide flexibility and dynamism.

A software life cycle model depicts the significant phases or activities of a software project from conception until the product is retired. It specifies the relationships between project phases, including transition criteria, feedback mechanisms, milestones, baselines, reviews, and deliverables. Software life cycle models describe the interrelationships between software development phases. The common life cycle models are (Bezroukov, 2009):

Waterfall Model

The Waterfall Model is the oldest and most well-known SDLC model. It involves a sequential step-by-step process from requirements analysis to maintenance (Purcell, 2007). The Waterfall model has many advantages including well-defined and understood requirements, an often times there ample time is set aside for the project. The main disadvantage of Waterfall model is its lack of flexibility. After project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements change or more information becomes available to the project team. During a normal development process requirements almost always change and the rigidity of the Waterfall model often leads to an implementation of the product that is obsolete as it goes into production.

The Waterfall Model should not be used for software development projects where requirements are not well-known or understood by the development team, the risk the project will fail is high. Additionally, not all the errors/problems related to a phase are resolved during the same phase. There is a tendency to push the risk down the line and this result in a situation where major part of the risk happens or rises only towards the end of the project, especially during the implementation phase, where the cost to rectify these risks also rises accordingly.

Spiral Model

In the Spiral SDLC Model, the development team starts with a small set of requirements and goes through each development phase (except Installation and Maintenance) for those set of requirements. Based on lessons learned from the initial iteration (via a risk analysis process), the development team adds functionality for additional requirements in ever-increasing "spirals" until the application is ready for the Installation and Maintenance phase

(production). Each of the iterations prior to the production version is a prototype of the application (Purcell, 2007).

The Spiral model was developed to respond to the limitations encountered in the Waterfall model by introducing a formal way to make changes to the project as requirements change. The advantages of the spiral model speak to the approach's ability to lead to continuous refinement. Specifically, the iterative approach used in this model allows development to begin even when all the system requirements are not known or understood by the development team. User feedback is used to make sure the project remains on track. The risk analysis step provides a formal method to ensure the project stays on track even if requirements do change. If new techniques or business requirements make the project unnecessary, it can be canceled before too many resources are wasted (Purcell, 2007).

The implementation of a project using the Spiral model requires highly skilled people in the many areas such as planning, risk analysis and mitigation, development, customer relation. Often times the process needs to be iterated more than once in order to arrive at the best solution. This can make the process more time consuming and somehow expensive.

Top-Down Model

In the Top-down SDLC model high-level requirements are documented, and programs are built to meet these requirements. Then, the next level is designed and built (Purcell, 2007). The advantage of the Top-Down model is the ability to focus on the big picture without getting tied down by a specific detail of the implementation. By successfully designing the major components at a high level, the hope is that detail implementation will become routine or seamless as thing fall into place. A major problem with

the Top-down model is that real system functionality is not added and cannot be tested until late in the development process. If problems are not detected early in the project, they can be costly to remedy later.

Bottom-Up Model

In the Bottom-Up SDLC model, the lowest level of functionality is designed and programmed first, and finally all the pieces are integrated together into the finished application (Purcell, 2007). The main advantage of the Bottom-Up model is that the most complex components are developed and tested first. There is a level of assurance that the functionality at the lowest level work correctly prior to constructing the overall system. The Bottom-up model also encourages the development and use of reusable software components that can be used multiple times across many software development projects. The problem with the Bottom-Up model is that there is no assurance that the working components will work together correctly in the finished system. For this to be the case extreme amount of coordination is required to make sure that the inputs and outputs of each component satisfies the needs of the adjoining component. Lack of coordination can lead to a failed system where the individual working components are unable to provide the services necessary for the overall system to be function correctly.

Hybrid Model

The Hybrid SDLC model combines aspects of the top-down and bottom-up models in order to use the advantages of both models and eliminate the disadvantages of each. This approach allows the development team to make changes to the system early in the project if problems occur with the high-risk components. Many of the SDLC models are a variation of the Hybrid Model.

Rapid Prototyping

The main idea of this model is to use prototypes built quickly to present to the application users as a starting point for an iterative development process. The Rapid Prototyping model is used for graphical user interface (GUI) applications such as web-based applications (Purcell, 2007). This model is covered in greater detail in Chapter 4 as it is the central aspect of the Rapid Application Development methodology.

Other Models

Other SDLC models include Model Driven Development, Chaos Model, Agile Programming Model, and many others. One significant trend in the development of new SDLC models is the integration of software design tools into the programming environment (Purcell, 2007). Software development tools such as Visual Studio are using prepackaged code modules saved in classes that programmers can reference. These classes contain functionalities such as databases connection, and login forms that programmers had to build for themselves in years past.

2.5 Future Trends

Software engineering is a relatively young field that is evolving at an exponential pace with projected future developments that many of us would have considered as science fiction only a few years ago. A lot of the growth can be attributed to the contribution of the internet which has expanded the horizon of development possibilities. There are many trends that could change the face of business in the upcoming years, and in most cases software development in playing a crucial part in the realization. One such trend is Cloud Computing. Cloud Computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS),

Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) (Cross-site scripting, 2004). Saas is one type of Cloud Computing that is directly affecting Software Development. This type of Cloud Computing delivers a single application through the browser to thousands of customers using a multitenant architecture. On the customer side, it means no upfront investment in servers or software licensing; on the provider side, with just one app to maintain, costs are low compared to conventional hosting (Eric Knorr, 2010).

As an integral part of the business environment, it is critical to ensure the software that we use is protected from any intrusion and disruptions. The increasing dependence on software to get critical jobs done means that software's value no longer lies solely in its ability to enhance or sustain productivity and efficiency. Instead, its value also derives from its ability to continue operating dependably even in the face of events that threaten it. The ability to trust that software will remain dependable under all circumstances, with a justified level of confidence, is the objective of software assurance. Software assurance has become critical because dramatic increases in business and mission risks are now known to be attributable to exploitable software (McGraw, 2006). We will examine way to ensure that the software that we build continue to function as intended despite the many risks both internal as well as external to the SDLC.

CHAPTER THREE

SOFTWARE SECURITY EXPLOITS

Gartner report estimates that it costs about $1 million a year on average for a company with 2,500 to 3,000 machines to patch its software and according to David Rice, former cryptographer for the NSA and Navy, author of Geekonomics: The Real Cost of Insecure Software, the total economic cost of security flaws in software is around 180 billion dollars a year in the U.S (Greenberg, 2008). It is therefore imperative to make sure that the software that is brought to market is reliable and secure. One way of ensuring security is to understand how software can be exploited and to put in place mechanisms to avoid those pitfalls.

When developing an application it is important to have an understanding of the attacker's point of view and to understand the limitations of your system. An understanding of the attacker's tactics allows the developer to implement effective countermeasures that will make it more challenging for the attacker to compromise your system. This chapter examines the common ways that attackers successful penetrate systems and provides an analysis of software security by examining threats, vulnerabilities, and attacks.

3.1 Threats to Software Security

Computer systems are constantly under threat, the software that allows organizations to function more effectively have also become the point of attacks. The threat to software security can be categorized into two groups:

1. **Threats during development *(mainly insider threats):*** A software engineer can sabotage the software at any point in its development life cycle through intentional exclusions from, inclusions in, or

modifications of the requirements specification, the threat models, the design documents, the source code, the assembly and integration framework, the test cases and test results, or the installation and configuration instructions and tools (McGraw, 2006).

2. **Threats during operation (both insider and external threats):** Any software system that runs on a network-connected platform is likely to have its vulnerabilities exposed to attackers during its operation. Attacks may take advantage of publicly known but unpatched vulnerabilities, leading to memory corruption, execution of arbitrary exploit scripts, remote code execution, and buffer overflows. Software flaws can be exploited to install spyware, adware, and other malware on users' systems that can lie dormant until it is triggered to execute (McGraw, 2006).

In order for a threat to become a danger for an application, there must be vulnerability in the system that it can exploit. Table 3.1 shows some common software vulnerabilities and the threats that are associated with them. During the last few years, the number of vulnerabilities being discovered in applications is far greater than the number of vulnerabilities discovered in operating systems. As a result, more exploitation attempts are recorded on application programs. The most "popular" applications for exploitation tend to change over time since the rationale for targeting a particular application often depends on factors like prevalence or the inability to effectively patch (Top Cyber Security Risks , 2009).

Table 3.1 Threats by Application Vulnerability Category (Source: (J.D. Meier, 2003))

Category	Threats
Input validation	Buffer overflow; cross-site scripting; SQL injection; canonicalization
Authentication	Network eavesdropping; brute force attacks;

	dictionary attacks; cookie replay; credential theft
Authorization	Elevation of privilege; disclosure of confidential data; data tampering; luring attacks
Configuration management	Unauthorized access to administration interfaces; unauthorized access to configuration stores; retrieval of clear text configuration data; lack of individual accountability; over-privileged process and service accounts
Sensitive data	Access sensitive data in storage; network eavesdropping; data tampering
Session management	Session hijacking; session replay; man in the middle
Cryptography	Poor key generation or key management; weak or custom encryption
Parameter manipulation	Query string manipulation; form field manipulation; cookie manipulation; HTTP header manipulation
Exception management	Information disclosure; denial of service
Auditing and logging	User denies performing an operation; attacker exploits an application without trace; attacker covers his or her tracks

The top 10 information security threats for 2010 according to Perimeter E-Security are: Malware, Malicious insiders, Exploited vulnerabilities, Careless employees, Mobile devices, Social networking, Social engineering, Zero-day exploits, Cloud computing security threats, and Cyber espionage (Top 10 information security threats for 2010, 2010). In the following section we are going to examine some of these threats in order to get an understanding of how they are used to compromise a system. Web applications in particular are often subjected to the threats highlighted above. Attacks against web applications constitute more than 60% of the total attack attempts observed on the Internet. These vulnerabilities are being exploited widely to convert trusted web sites into malicious websites serving content that contains client-side exploits. Web application vulnerabilities such as SQL injection and Cross-Site Scripting flaws in open-source as well as custom-built applications account for more than 80% of the vulnerabilities being discovered (Top

Cyber Security Risks , 2009). Table 3.2 shows the ten most common application vulnerabilities detected on users' computers.

Table 3.2. Ten most common vulnerabilities detected on users' computers (Source: Namestnikov, 2010)

№	Secunia ID	Change	Vulnerability	Impact	Percentage of users affected	Release date	Rating
1	SA 38805	7	Microsoft Office Excel Multiple Vulnerabilities	System access, execution of arbitrary code with local user privileges	39.45%	2009-06-09	Highly Critical
2	SA 37255	new	Sun Java JDK / JRE Multiple Vulnerabilities	Security bypass	38.32%	2010-02-12	Highly Critical
3	SA 35377	-2	Microsoft Office Word Two Vulnerabilities	System access, execution of arbitrary code with local user privileges	35.91%	2010-03-09	Highly Critical
4	SA 38547	-1	Adobe Flash Player Domain Sandbox Bypass Vulnerability	Security bypass	30.46%	2009-04-03	Moderately Critical
5	SA 31744	1	Microsoft Office OneNote URI Handling Vulnerability	System access, execution of arbitrary code with local user privileges	27.22%	2007-01-09	Highly Critical
6	SA 34572	-2	Microsoft PowerPoint OutlineTextRefAtom Parsing Vulnerability	System access, execution of arbitrary code with local user privileges	21.14%	2008-09-09	Extremely Critical
7	SA 39272	new	Adobe Reader / Acrobat Multiple Vulnerabilities	System access, execution of arbitrary code with	21.12%	2010-04-04	Highly Critical

				local user privileges Cross-site scripting			
8	SA 29320	2	Microsoft Outlook "mailto:" URI Handling Vulnerability	System access, execution of arbitrary code with local user privileges	19.54%	2008-03-11	Highly Critical
9	SA 39375	new	Microsoft Office Publisher File Parsing Buffer Overflow Vulnerability	System access, execution of arbitrary code with local user privileges	16.08%	2010-04-13	Highly Critical
	SA 37690	-1	Adobe Reader/Acrobat Multiple Vulnerabilities	System access, execution of arbitrary code with local user privileges Cross-site scripting	15.57		

SQL Injection

SQL Injection is an attack technique used to exploit web sites by altering backend SQL statements through manipulating application input (Web Security Glossary, 2004). SQL Injection happens when a developer accepts user input that is directly placed into a SQL Statement and doesn't properly filter out dangerous characters. This can allow an attacker to not only steal data from your database, but also modify and delete it. Certain SQL Servers such as Microsoft SQL Server contain Stored and Extended Procedures (database server functions). If an attacker can obtain access to these Procedures it may be possible to compromise the entire machine. Attackers commonly insert single quotes into a URL's query string, or into a forms input field to test for SQL Injection. SQL Injection on the internet can more

or less be divided into two sub-categories: Legitimate SQL Injection and Malicious SQL Injection. Many web applications on the Internet still use "SQL Injection" for their normal functionality. The web applications that legitimately use SQL Injection are guaranteed to be vulnerable to the tools and techniques used by attackers to perform Malicious SQL Injections. The servers that house these applications may have a higher compromise rate not only because they are known to be vulnerable, but also because they need to distinguish between legitimate and malicious injects to identify attacks (Top Cyber Security Risks , 2009).

Cross-Site Scripting

Cross-site scripting (XSS) is a security exploit in which the attacker inserts malicious coding into a link that appears to be from a trustworthy source. When someone clicks on the link, the embedded programming is submitted as part of the client's Web request and can execute on the user's computer, typically allowing the attacker to steal information.

Web forms that dynamically return an error message including user input data make it possible for attackers to alter the HTML that controls the behavior of the form and/or the page. Attackers do this in a number of ways, for example by inserting coding into a link in a forum message or in a spam message. The attacker may use e-mail spoofing to pretend to be a trusted source.

Like other Web-based exploits, such as SQL injection, much of the blame for cross-site scripting is placed on the insecure applications that make it possible. Web server applications that generate pages dynamically are vulnerable to a cross-site scripting exploit if they fail to validate user input and to ensure that pages generated are properly encoded. A vulnerability that enables cross-site scripting is sometimes referred to as an XSS hole.

To protect against cross-site scripting, experts recommend that Web applications should include appropriate security mechanisms and servers should validate input as a matter of course (cross-site scripting, 2004).

Zero-Day Vulnerability

A zero-day vulnerability occurs when a flaw in software code is discovered and code exploiting the flaw appears before a fix or patch is available. Once a working exploit of the vulnerability has been released into the wild, users of the affected software will continue to be compromised until a software patch is available or some form of mitigation is taken by the user.
The "File Format Vulnerabilities" continue to be the first choice for attackers to conduct zero-day and targeted attacks. Most of the attacks continue to target Adobe PDF, Flash Player and Microsoft Office Suite (PowerPoint, Excel and Word) software. Multiple publicly available "fuzzing" frameworks make it easier to find these flaws. The vulnerabilities are often found in 3rd party add-ons to these popular and wide-spread software suites, making the patching process more complex and increasing their potential value to attackers. There is a heightened risk from cyber criminals, who can discover zero-day vulnerabilities and exploit them for profit (Top Cyber Security Risks , 2009).

Buffer Overflow

A Buffer Overflow is a flaw that occurs when more data is written to a block of memory, or buffer, than the buffer is allocated to hold. Exploiting a buffer overflow allows an attacker to modify portions of the target process' address space. The attacker's goal is almost always to control the target process' execution. This is accomplished by identifying a function pointer in memory that can be modified, directly or indirectly, using the overflow. When such a pointer is used by the program to direct program execution through a jump

or call instruction, the attacker-supplied instruction location will be used, thereby allowing the attacker to control the process.

Buffer Overflows can be categorized according to the location of the buffer in question, a key consideration when formulating an exploit. The two main types are Stack-Based Overflow and Heap-Based Overflow. Buffers can be located in other areas of process memory, though such flaws are not as common (Auger, 2010).

Buffer overflow has become one of the preferred attack methods for writers of viruses and Trojan horse programs. Crackers are adept at finding programs where they can overfill buffers and trigger specific actions running under root privilege -- say, telling the computer to damage files, change data, disclose sensitive information or create a trapdoor access point (Kay, 2003).

Email Attacks

Waves of targeted email attacks, often called spear phishing, are exploiting client-side vulnerabilities in commonly used programs such as Adobe PDF Reader, QuickTime, Adobe Flash and Microsoft Office. This is currently the primary initial infection vector used to compromise computers that have Internet access. Those same client-side vulnerabilities are exploited by attackers when users visit infected web sites. Because the visitors feel safe downloading documents from the trusted sites, they are easily fooled into opening documents and music and video that exploit client-side vulnerabilities. Some exploits do not even require the user to open documents. Simply accessing an infected website is all that is needed to compromise the client software. The victims' infected computers are then used to propagate the infection and compromise other internal computers and sensitive servers incorrectly thought to be protected from unauthorized access by external entities. In many cases, the ultimate goal of the attacker

is to steal data from the target organizations and also to install back doors through which the attackers can return for further exploitation. On average, major organizations take at least twice as long to patch client-side vulnerabilities as they take to patch operating system vulnerabilities (Top Cyber Security Risks , 2009).

3.2 Improving Software Security

It is impossible to build defect-free software no matter how security-conscious we are and no matter our expertise. The idea of software security is to minimize the risks associated with the use of the particular software. In order to win in this constant ongoing battle, project teams must integrate security into the development of the software and put in place mechanisms to continually improve and respond to threats as they come. Secure software is software designed with security in mind, developed with security controls and deployed in a secure state. While it might have the potential of being breached, the repercussion of a breach is greatly diminished. Secure design and architecture, secure development with security controls built in by default, and secure deployment or release, all work together to minimize the impact of software vulnerability that gets exploited (Paul, Software assurance: embedding security into the software development lifecycle.(SOFTWARE WORLD INTELLIGENCE), 2008)

A number of factors influence how likely software is to be secure. For instance, software vulnerabilities can originate in the processes and practices used in its creation. These sources include the decisions made by software engineers, the flaws they introduce in specification and design, and the faults and other defects they include in developed code, inadvertently or intentionally. Other factors may include the choice of programming languages and development tools used to develop the software, and the configuration and behavior of software components in their development and

operational environments (McGraw, 2006). Software developers must be trained to use and adhere to correct security development and coding standards. They must be able to understand the possible vulnerabilities and the risks they present for the organization.

Software that has been developed with security in mind generally reflects the following properties throughout its development life cycle:

1. **Predictable execution:** There is justifiable confidence that the software, when executed, functions as intended. The ability of malicious input to alter the execution or outcome in a way favorable to the attacker is significantly reduced or eliminated.

2. **Trustworthiness**: The number of exploitable vulnerabilities is intentionally minimized to the greatest extent possible. The goal is no exploitable vulnerabilities.

3. **Conformance:** Planned, systematic, and multidisciplinary activities ensure that software components, products, and systems conform to requirements and applicable standards and procedures for specified uses (McGraw, 2006).

Understanding how the SDLC process works allows us to put in place the necessary safeguards to ensure that vulnerabilities are minimized and each threat is accounted for. For the SDLC process to be secure, security related activities must be incorporated into each phases during the development of the application. These activities must be based on the security needs of the organization as defined in the project requirements. The goal of each security activity must coincide with the objectives of the CIA triangle. The development team should use all security controls that can improve the confidentiality, integrity, and availability. These controls can be

administrative controls, physical controls, or technical controls. For example providing security training of software developers can be an effective administrative control. When developers are trained correctly, they will more likely to make good design decision from the start therefore reducing the vulnerabilities.

Testing can be an effective control and it must be integrated in the SDLC in order to improve security. Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test-driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed (Software testing, 2010).

The organization must develop policies and standards and communicate these to each project team member and have them pledge that they will adhere to the standard. Policies, standards, and procedures should be formulated to address software development methodology and establish practical built-in security features. Some examples include identification and authentication policy, remote access policy, use of company resources policy, software security standards, data classification standards, encryption standards, logging and monitoring standards and disaster recovery and business continuity standards. These policies must be enabled by processes that span the development lifecycle and ensure continuity of security measures, while all must be managed by people with apt security skills (Paul, Software assurance: embedding security into the software development lifecycle.(SOFTWARE WORLD INTELLIGENCE), 2008).

It is only by following the above prescription that software can be considered reliable and secure. Software security cannot be accidental; it must be

intentional. The most critical difference between secure software and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software (McGraw, 2006).

CHAPTER FOUR

THE SECURE RAPID APPLICATION DEVELOPMENT METHODOLOGY (S-RAD)

RAD (rapid application development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that defers design improvements to the next product version
- Less formality in reviews and other team communication

(What is rapid application development, 2010)

Having gained a deeper understanding of the Software development process and the security challenges that development teams face, we are now ready to jump into the heart of our project. This chapter will focus on the presenting the traditional RAD approach in detail and introducing ways to improve it. I will present a new variation of Rapid Application Development which I'm calling S-RAD or Secure Rapid Application Development that makes security an integral part of the development process.

4.1 Overview

The accelerating pace and cost-reduction of software development driven by business pressures is greatly impacting the development process and subsequently the end product. In a recent ExecutiveBrief Software Development Trends Survey of more than 500 senior-level business leaders and software development professionals, when asked about their top software development priorities for 2010, one-third of respondents identified

reducing operational cost and expenses among their top priorities. It was also evident from this survey that methodologies that fall in the agile category were becoming more prominent. The Agile approach increased its dominance as the software development methodology of choice with over one-half selecting its use as their preferred method, notably up from 42% in 2009. Iterative was the second most selected methodology with 13% giving it their top choice (ExecutiveBrief Software Development Trends Survey Indicates Promising Business Environment for 2010, 2010). The oldest and best known agile methodology is the Rapid Application Development (RAD) which is based on the rapid prototyping model of the SDLC. RAD is a linear sequential software development process model that emphasizes an extremely short development cycle using a component-based construction approach. If the requirements are well understood and defined, and the project scope is constrained, the RAD process enables a development team to create a fully functional system within very short time period often times without compromising usability, features, and/or execution speed (Ravindran, 2009). RAD involves constant evaluation and feedback in an iterative loop aimed at improving the quality of the product.

4.2 Phases of RAD

The RAD model has the following phases:

1. **Requirement Planning Phase**: This is the stage where the objectives, functionality, and scope are established. In this step the development team meets to develop a high level list of initial requirements as well as determine the project scope. At the conclusion of this step the development team will have a clearer picture of what information is generated, who generates it, and who process. This

information is used to develop the initial list of features and the way they should function as a unit.

2. **User Design Phase:** In this step the development team which is comprised of the main stakeholders meets to plan how the essential parts of the system should work. The end result of this step is a design document depicting layouts of the system as well as business rules, test plans.

3. **Construction Phase:** In this step the prototype is converted into a functional application. At this stage actual coding occurs, the application developers add the functionalities to the prototype. This is done in an iterative cycles of development, testing, requirements refining, and development again, until the application is complete.

4. **Cutover Phase:** In this stage the final user testing and training is done and decisions are made on the publication of the application system. This step involves a review of the constructed system by the stakeholders to determine whether it meets their expectations. Features that meet expectations are sent for publication whereas for features that fall short of expectations are reentered into an iterative design loop.

4.3 Foundation of RAD

The fundamental elements RAD that distinguish it from other methodologies are prototyping, iterative development, time boxing, team members, management approach, and RAD tools.

Prototyping

One the earlier steps in the RAD methodology is the development of a prototype system that is used by the development team to refine the requirement. The initial prototype is usually a simplified version of the finished product that is put together quickly using Computer Aided Software Engineering CASE tools. The focus of the prototype is to convert requirement into a model.

Iterative Development

Iterative development is the process of adding functionality to a system at each cycle. RAD uses short development cycles that allow the client to review the design and make recommendations for the next cycle. The process is repeated until all functionality has been developed.

Time Boxing

Time boxing is the process of putting off features to future application versions in order to complete the current version in as short amount of time as possible. Time boxing is used to help the application development process move along when there is a blockage.

Team Members

Successful implementation of RAD requires a small team of highly skilled, motivated, and versatile members. The team should consist of no more than six members including the client. The development process requires a high level of communication between the group members who collaborate often in workshop sessions. Team members should ideally have experience in Rapid Application Development and should have experience with the Computer Aided Software Engineering tools.

Management

The role of management is extremely important to a successful RAD implementation. Management should be the driving force behind the process by creating the necessary environment for the successful implementation of RAD. The RAD methodology requires quick decisions in order to prod the process along and to overcome the pitfalls that are common in lengthened development cycles. Management's role also includes clearing client misunderstandings and making sure that the project remains on target.

RAD Tools

Technology plays a big role in the successful implementation of the RAD model. Because of the limited time and resources, the RAD methodology makes use of technological tools to speed development there are several tools available in the market today and they can generally be classified as:

1. Data Integration tools:
2. Development Environments:
3. Requirements Gathering Tools:
4. Data Modeling Tools:
5. Code Generation Tools:

4.4 Appropriate RAD Projects

The success of a project greatly depends on the choice of the model. Rapid Application Development is not appropriate for all projects. Successful RAD projects have the following characteristics:

1. Small Scope
2. Modular in nature
3. Small and experienced project team
4. Involvement of end user

5. Strong management support
6. Valuable reason to speed up the process
7. Adequate development tools

RAD is not suitable for projects dealing with highly sensitive and complex operations. For example an application that manages the electric grid or nuclear power system for a town should not be implemented using RAD because of the complexity involved cannot be managed in a short high speed design process. These types of systems require deliberate well thought out design. Systems that can benefit from RAD include e-commerce web sites, inventory management applications, and content publishers such as newspaper websites.

4.5 Benefits of RAD

The RAD methodology presents many benefits when used in the right situation. Projects implemented using RAD benefit from faster speed and higher quality because users are involved throughout the process. Users are able to see tangible proof of the product during the development and can participate in improving it before the completion of the project. Often times the process can produce 80% of the solution in only 20% of the time needed to produce a total whole solution using other models. This is due in part to Time Boxing which makes it possible to satisfy the most useful business requirements in a timely manner. Additionally components built using RAD can be reused in other applications due to their modular nature.

4.6 Disadvantages of RAD

The RAD methodology's focus on building the systems as fast as possible makes it difficult to plan for the future. Features are often not easily scalable because the focus is on doing what is needed now. In order to meet

deadlines some features might not be implemented, therefore users might end up with a product that is less than expected. In order The RAD methodology to succeed the design team must be made of highly skilled and most likely highly paid members which imply higher cost. Any changes in personnel during the process can compromise the project as additional time will be needed to find and bring the replacement up to speed. The amount of time would be negatively affected by the fact that RAD projects are often loosely documented

4.7 Securing RAD

Integrating security activities in the RAD model requires a great balancing act. While the intention is to improve the security posture, the potential increase of the development time and budget that could result from the activities need be the dealt with carefully. The measures must not significantly alter the main objectives and benefits of the RAD model which center on reducing development time, cost, and improve quality.

Proactively tackling software security is often under-budgeted and dismissed as a luxury. In an attempt to shorten development schedules or decrease costs, software project managers often reduce the time spent on secure software practices during requirements analysis and design (McGraw, 2006). In addition, they often try to compress the testing schedule or reduce the level of effort. Skimping on software quality is one of the worst decisions an organization that wants to maximize development speed can make; higher quality (in the form of lower defect rates) and reduced development time go hand in hand. Figure 4.1 illustrates the relationship between defect rate and development time.

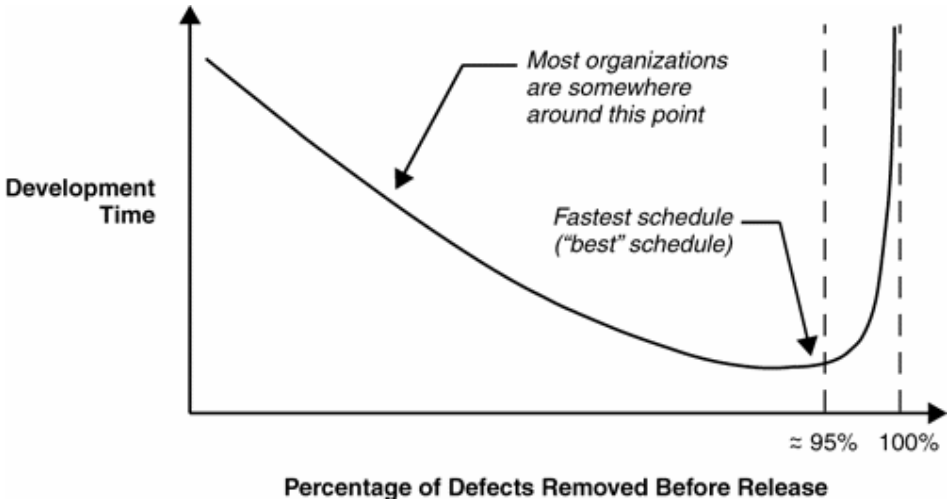

Figure 4.1 Percentage of Defects Removed Before Release (Source: McGraw, 2006

The major processes that reduce security defects are secure design, threat modeling, Static code analysis, and security testing

Secure design: Properly implementing security features, to minimize and harden the attack surface, and layer multiple security mechanisms into the design (Wysopal, 208). Secure design begins with a voluntary effort and acknowledgement of the need for security. As part of the design process, a determination must be made as to what aspects of the system require protection and the level of protection required. The designer must determine the nature of the data flowing through the system and all the components that interact with the data as part of a normal process. This determination would lead to the establishment of what data needs to be secured and what rights a specific user can have on the data. The objectives of the design must be ensuring the confidentiality, integrity, and availability of the data flowing through the system. The design must ensure that the critical data and resources are concealed from people that do not have the rights to them; it must also ensure that the data and

resources are trustworthy, and can be accessed by users with the right privileges whenever they need it.

Secure design must be integrated into the culture of the organization. This requires a clear mandate from the top plus means to train members of the team in secure requirement gathering and design practices must be provided. This training should be extended to business analysts who help define the application, developers who design and develop it, testers who look for defects, help desk workers who talk to the application users and systems administrators who monitor the application. Developers, testers and program managers must learn to understand the importance of security issues, communicate on those issues with others on the project, and take positive, proactive steps to reduce risks endemic to software applications. Standard baseline security checklists can be quite useful in this phase, and might address this point by incorporating rules such as, “sensitive data including Social Security Numbers must not be used in ways that might lead to accidental exposure of that data,” or “SSN must always be encrypted when stored in a database to safeguard against accidental exposure (Secure By Design: Security in the Software Development Lifecycle, 2005).

Threat modeling: Performed to determine whether the design mitigates the risks posed by the software's functionality. Threat modeling allows the designers to understand the attacker's point of view. If the threat isn't properly mitigated, the design must be changed or the risk assumed (Wysopal, 208). Threat modeling encompasses a review of the design to determine where a potential

threat might exploit the system. A review of all entry points into the system, assets, threats, and dependencies is performed in order to gage whether the design successfully mitigates the risks.

Static code analysis: This involves reviewing the code to ensure that it is structured according to accepted industry standards. In years past it involved manual review of each line of code by experts in the programming field. It can now be accomplished using analysis tool with little or no intervention from a human being. There are many analysis tools in the market today, commercial and open source, which are used to analyze code. Tools such as Rational Software Analyzer and Fortify can analyze code from multiple languages whereas others such as NDepend (.Net) focus on a particular language or family.

The results of the static analysis are triaged for security flaws that will truly impact the security of the software. The triaged flaws then should be entered into the defect-tracking system and treated as "must-fix" bugs (Wysopal, 208). In order to ensure that the must-fix bugs are resolved, the project manager can enter them into a tool such as Numara track-it which is a help desk software that is used to keep track of projects and tasks. Using Track-it, each bug would be assigned to a specific developer to resolve within a prescribed amount of time frame.

Security testing: Security testing is similar to standard software testing except, instead of verifying that the functionality of the program works as intended, the testing is attempting to get the software to perform functions it wasn't designed to do (Wysopal, 208).

To perform security testing, the tester must have an understanding of the common vulnerabilities and the methods that attacks usually occur. Security testing should be based on the risks identified during system analysis with the objective being to push the system to the limit to see if it will break. Unlike functional testing, it is often difficult to assess the success of security testing because it is nearly impossible to account for all situations that might happen.

Despite the time constraints of the RAD model it is possible to incorporate all four processes mentioned above and still maintain the required speed. The key is to make sure that security is part of the design and simple but efficient security steps are performed as embedded part of each phases of the RAD methodology. The tables 4.1, 4.2, 4.3, and 4.4 highlight the main security activities that must be incorporated in each phase of RAD in order to improve the security posture of the application being developed. The development team should have security discussions throughout the process in order to make sure that everyone involved from management down to the user has a solid understanding of the security objectives. The result of embedding the new security activities into the traditional RAD methodology is the creation of S-RAD or Secure Rapid Application Development methodology.

Table 4.1 Requirement Planning Security Activities

Security Activity	Description
Conduct the initial risk assessment	This is a qualitative high level assessment in order to get a better understanding of the vulnerabilities and treats that you are likely to encounter.
Information Classification	From the requirements and risk analysis, make a list of the data that you will need to

	transmit, store and assign the level of sensitivity to each such as high, moderate, medium, and low.
Determination of any privacy requirements	Legal and industry privacy requirements should be incorporated in the design in order to ensure that the project
Develop Security Test Plan	A plan detailing the area to test and how to test.

Table 4.2 User Design Security Activities

Security Activity	Description
Select Security Controls	Judging from the risk assessment, determine the appropriate controls to mitigate the risks.
Complete Security Test Plan	Revise the test plan to account for the security measures that have been put in place. The plan should include a way to test the effectiveness of the controls.

Table 4.3 Construction Security Activities

Security Activity	Description
Perform functional and security testing	Test each module to make sure that it works correctly and that the controls that are put in place mitigate the treats.
Conduct a risk assessment	Evaluate the risks remaining after the test has been conducted.
Update Security Controls	Update security controls by reinforcing those that working and changing those that are not working.

Table 4.4 Cut Over Security Activities

Security Activity	Description
Integrate the information application into its environment	Perform Security tests on the application in the integrated environment to determine if the controls remain effective.
Update security controls	Update security controls by reinforcing those that working and changing those that are not working.
Review deployment status	Analyze the security posture of the application to determine which components can be published in this iteration and which should be timeboxed for further testing.

It is essential to introduce the security activities during development because they help the development team to discover defects much earlier than they would otherwise. As soon as software is deployed and enters the operational/maintenance phase, it immediately becomes a potential target for attacks and many organizations only implement security well after the application has been deployed as a response to some threat or after vulnerability has been exploited. Incorporating security early, and maintaining it throughout all the different phases of the SDLC, has been proven to result in less expensive and more effective security than adding it to an operational system. Studies show that the relative cost of fixing defects in production is 30 to 100 times more expensive. Figure 4.2 depicts the results of a study conducted by IBM Systems Sciences Institute on the relative cost of fixing defects (Paul, The Need for Secure Software, 2009).

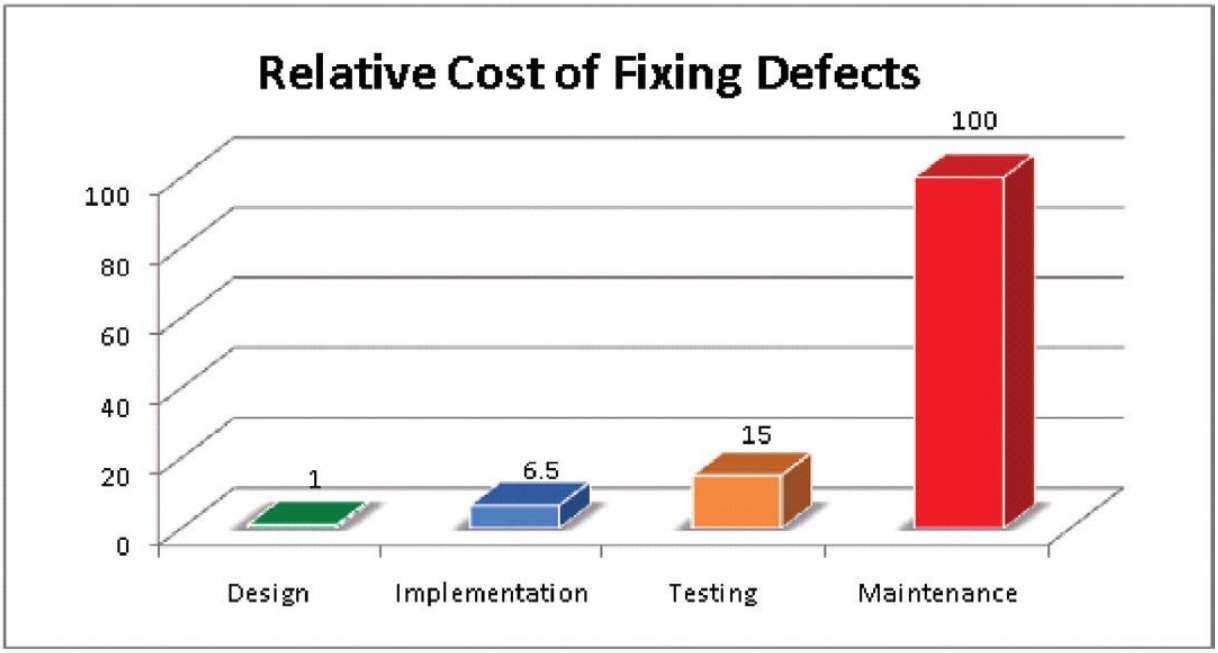

Figure 4.2: Investing in security early can dramatically decrease the cost of fixing defects (Source: (Paul, The Need for Secure Software, 2009))

The effect of software security breaches and data loss can threaten the survival of an organization as a result of damaged reputation and devastating fines. In order to ensure security, organizations must address software security through the entire lifecycle, from initiation to disposal. Attempting to resolve security shortcomings in the production environment with a patch-and-release cycle, as is often the case today, does not address the root of the problem.

The activities described above should not take too long in order to keep the process flowing. In order for this to be true, any organization planning to implement the RAD methodology should have general testing policies in place prior to engaging in the development process.

4.8 RAD in Action - Case Study

In this section we will examine a fictional case study example involving two printing companies in order to illustrate how S-RAD should be implemented. The objective is show how security can be baked into RAD despite the limited time available to complete the project.

Waka Waka Publishing

Waka Publishing is a company that sells magazines. Over the last year, print magazine sales have decreased substantially, so they decided to put their content online and sell subscriptions for content and download. In order to keep their customer base informed of the new content, they have decided to institute an email alert system that will send customers alerts according to subjects and keywords of interest to them. During registration, users would be able to select which subjects and keywords are of interest to them. Leangue Publishing Company, which is Waka Waka's main competitor, has just launched a similar system and, as a result, is gaining more subscribers to their site, many of whom are former Waka Waka print subscribers. In an attempt to salvage the bottom line, Waka Waka CEO has mandated his team to come up with a solution within two months.

We are going to discuss security activities that Waka Waka publishing would have to integrate into their development process in order to ensure that their users' information is protected. But before talking about the security activities, we need to determine whether the project above is appropriate for RAD. A RAD project has the following characteristics:

- **Small Scope**: Since Waka Waka is already selling content in their site, the email alert system is only concerned with managing user emails and their subjects of preference. The goal is to tie the existing user with the keyword and subjects of interest.

- **Modular in nature**: The project can be developed as a standalone application, as it only requires user emails from the existing system.

- **Small and experienced project team**: Waka Waka' s team is experienced in developing applications for the print industry. We will work under the assumption that they have an experienced team.

- **Involvement of end user**: Many Waka Waka employees are also subscribers to their online print site and will be brought in during the development of the new application.

- **Strong management support:** The project has been initiated by the CEO himself and is of great importance for the company's bottom line. The CEO has put his top people on this project.

- **Valuable reason to speed up the process:** The Company is losing its competitive edge and is experiencing a decrease in revenues.

- **Adequate development tools:** The development team possesses all the tools required.

Requirement Planning

In this phase the team performs and initial risk assessment to determine which threats they could encounter in managing and transmitting user information. They also rank the sensitivity of information that they would need to manipulate. General security discussion is held, and a test plan is discussed.

Potential threats that could be encountered are assigning the wrong keyword and subjects to a user and exposing user personal information. Assigning the wrong keyword and subject to the user would result in sending them information that they are no interested in. Some users might consider this as junk mail and block the sender. Maintaining the confidentiality of the user

information is critical because some criminals might try to masquerade as the user in order to obtain sufficient information that would allow them to penetrate the system and gain access to more valuable information such as credit card numbers.

In order to develop a clear strategy, the development team institutes a simple security classification scheme that ranks the severity of threats in a range from one to five. The scheme determines that one is low, two is medium, three is moderate, four is high, and five is severe. Using this classification scheme, the threat of assigning the wrong keyword or password is rated as moderate because although such an error might cause an embarrassment for Waka Waka it can be corrected easily. Exposing user personal information is classified as severe because such an error can have implications beyond the relationship between the user and Waka Waka as an organization in the event that the information fall in the wrong hands.

User Design

Security activity in this step include selecting security controls to mitigate the risks highlighted in the requirement planning phase, and designing a test plan to verifies that the risks are handled appropriately. Controls that can be implemented to ensure that users only receive alerts for which they signed up for start with correct design. Designing a correct relationship between the user and the content is a key to ensuring that the systems work as intended. The design team should avoid creating a complex relationship model because complexity and security often conflict. They should also invest in a strong authentication system perhaps a two-step authentication in order to protect against criminals masquerading as legitimate users. In addition to the traditional user Id and password, the user would be required to supply another piece of information such as a private pin number in order gain

access to the system. These increases the chances that a hacker would not be able easily guess the log in information. The testing plan should focus on penetration testing to see how the system handles attempted intrusion.

Construction

Security activities in this stage revolve around implementing the test plan and revising the controls as threats are mitigated and new ones are discovered during construction. Each module must be tested independently to ensure that the design meets the security objectives. Activities such as creating test user account and assigning keywords and content and running tests to make sure that the users receive only the appropriate alerts. Other activities should involve attempting to break the authentication and system and masquerading as existing users.

Security testing activities for the Waka Waka application will include the following steps:

Functional Tests

- Ensure that users can login using the correct credentials
- Ensure that users can sign up for alerts
- Ensure that, if a user has an alert, an email with the correct message is sent to them

Logical Tests

Authentication

- Ensure that the user cannot sign up for an alert unless they are logged in.
- Ensure that users cannot create anonymous accounts

Login

- Ensure that user can only log in using their correct credentials
- When a user attempts to login using incorrect information, ensure that only a generic message is displayed and not one that is too specific so that you do leak information about existing user accounts.

Email

- Ensure that sensitive user information such as social security numbers and credit card numbers are not included in the alert email.

Regression Test

- Retest previous security issues to make sure that new changes do not cause them to resurface.

The programmers are primarily responsible for making sure that the security testing is done because security testing requires a certain level of experience and knowledge. Once the programmers have completed their part, they hand over the testing to a selected group of users in order get an unbiased view of the application.

Cut Over

In this phase, the different components are connected to one another and tests are performed to make sure that the integration does not create more vulnerabilities. When new threats are discovered, appropriate controls are put in place to mitigate the risk. The user authentication system, the system that determines which user receives which alert, and the component in charge of sending email to the user are connected and tested as a unit.

This would involve creating several user accounts and assigning alerts to each. The type of testing will mostly involve releasing a Beta version of the application to a selected group of users to test over several days.

SUMMARY

Securing an organization requires an understanding of your assets, because you cannot secure what you don't know. Many organizations are just now beginning to understand the need to implement security and often times it only happens after they have been victimized in one way or another. It is of great importance to address security and do it within the framework of an organization's normal business process in order to prevent catastrophic and unplanned events from crippling the operation of your organization. Security is an ongoing process that must be kept up to date and improved upon. There is no such thing as perfect security. The best you can do is to keeping yourself informed on the issues and try to mitigate the risks.

Application Security has become an important component of the overall defense strategy for an organization because hackers have focused a lot of their attention on the vulnerabilities present in many enterprise software. Organizations are therefore encouraged to build a strong defense mechanism to ensure that their software does not become the weakest link in their overall enterprise defense. Many organizations have understood the need to integrate security earlier in their process, but there is still a lot of work to do. Defects in the software can become very costly for organizations if there is no framework in place to respond to the incident appropriately. It is commonly believed that the earlier a defect is found, the cheaper it is to fix it. As we saw earlier in Figure 4.2 on page 44, the cost of fixing the defect rises substantially as you move along the phases of development. For example, a defect found during the design phase will cost 15 times less that if it were found later on during the testing phase.

Rapid Application Development can be an effective methodology for building applications if appropriate security measures are embedded in the process.

Securing the RAD process does not necessarily require an investment in expensive tools or technology but rather in being conscious about the vulnerabilities and to address them effectively by embedding appropriate control where needed. RAD is only beneficial when implemented correctly. Incorrect implementation can result in the method becoming more time consuming than other approaches and difficult to secure.

Works Cited

History of software development. (1997). Retrieved from Bizymoms: http://www.bizymoms.com/computers-and-technology/software-development.html

cross-site scripting. (2004, 08 31). Retrieved from WhatIs.com: http://searchsoftwarequality.techtarget.com/definition/cross-site-scripting

Cross-site scripting. (2004, 08 31). Retrieved from WhatIs.com: http://searchsoftwarequality.techtarget.com/definition/cross-site-scripting

Web Security Glossary. (2004, 02 23). Retrieved from http://www.webappsec.org/projects/glossary/

(2005). *Secure By Design: Security in the Software Development Lifecycle.* Arctec Group.

What is SQL Injection. (2008). Retrieved from http://www.cgisecurity.com/questions/sql.shtml

Top Cyber Security Risks . (2009, 09). Retrieved from Sans: http://www.sans.org/top-cyber-security-risks/trends.php

(2010). *ExecutiveBrief Software Development Trends Survey Indicates Promising Business Environment for 2010.* ExecutiveBrief.

Software development. (2010, October 28). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Software_development

Software testing. (2010, November). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Software_testing

Software testing. (2010, November). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/Software_testing

Top 10 information security threats for 2010. (2010, January 14). Retrieved from Help Net Security: http://www.net-security.org/secworld.php?id=8709

What is rapid application development. (2010). Retrieved from Whatis: http://searchsoftwarequality.techtarget.com/sDefinition/0,,sid92_gci214246,00.html

Alberto Sillitti, X. W. (2010). *Agile Processes in Software Engineering and Extreme Programming.* Berlin: Spring-Verlag Berlin Heidelberg 2010.

Allen, J. (2007). Why is Security a Software Issue? . *EDPACS, Volume 36, Issue 1 July 2007*.

Auger, R. (2010, 06). *Buffer Overflow* . Retrieved from http://projects.webappsec.org/w/page/13246916/Buffer-Overflow

Bezroukov, D. N. (2009, August 12). *Software Life Cycle Models*. Retrieved from softpanorama: http://www.softpanorama.org/SE/software_life_cycle_models.shtml

Eric Knorr, G. G. (2010, 07 19). What cloud computing really means. *InfoWorld*.

Greenberg, A. (2008). A Tax On Buggy Software. *Forbes*.

J.D. Meier, A. M. (2003). Threats and Countermeasures. In J. Meier, *Improving Web Application Security.* Microsoft Corporation.

Kay, R. (2003). QuickStudy: Buffer Overflow. *Computerworld*.

Longstreet, D. (2006). *History of Software Productivity*. Retrieved from Softwaremetric: http://www.softwaremetrics.com/Articles/history.htm

McGraw, G. (2006). *Software Security: Building Security.* Boston, MA : Addison-Wesley.

Namestnikov, Y. (2010, August). *Information Security Threats in the Second Quarter of 2010*. Retrieved from Securelist.com.

Osterweil, L. J. (2007, March). A Future for Software Engineering.

Paul, M. (2008). Software assurance: embedding security into the software development lifecycle.(SOFTWARE WORLD INTELLIGENCE). *Software World*.

Paul, M. (2009). *The Need for Secure Software.* (ISC)².

Purcell, J. (2007). *Defining and Understanding Security in the Software Development Life Cycle.* The SANS Institute.

Ravindran, C. (2009, January 9). *Rapid Application Development Model (RAD)*. Retrieved from Ezine Articles: http://ezinearticles.com/?Rapid-Application-Development-Model-(RAD)&id=412312

Richard Kissel, K. S. (2008). *Security Considerations in the System Development Life Cycle.* National Institute of Standards and technology.

Thibodeau, P. (2002, June 25). Buggy software costs users, vendors nearly $60B annually. *Computerworld*, p. 2002 .

Wysopal, C. (208). Building security into your software-development lifecycle. *SC Magazine*.