Designing Security into Software

by

Chang Tony Zhang

M.S. Computer Science (1999)

University of New Hampshire

Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology

June 2006

Signature of Author [illegible]
Chang Tony Zhang
System Design and Management Program
May 2006

Certified by [illegible]
Michael A. Cusumano
Thesis Supervisor
Sloan Management Review Distinguished Professor of Management

Certified by [illegible]
Patrick Hale
Director
System Design and Management Program

Contents

Designing Security into Software

by

Chang Tony Zhang

Submitted to the System Design and Management Program
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Engineering and Management

Abstract

When people talk about software security, they usually refer to security applications such as antivirus software, firewalls and intrusion detection systems. There is little emphasis on the security in the software itself. Therefore the thesis sets out to investigate if we can develop secure software in the first place.

It first starts with a survey of the software security field, including the definition of software security, its current state and the research having been carried out in this aspect. Then the development processes of two products known for their security: Microsoft IIS 6.0 and Apache HTTP Web Server are examined. Although their approaches to tackle security are seemingly quite different, the analysis and comparisons identify they share a common framework to address the software security problem - designing security early into the software development lifecycle.

In the end the thesis gives recommendations as to how to design security into software development process based upon the principles from the research and the actual practices from the two cases. Finally it describes other remaining open issues in this field.

Thesis Supervisor: Michael A. Cusumano

Title: Sloan Management Review Distinguished Professor of Management

Acknowledgements

I would like to thank my thesis advisor, Professor Michael Cusumano, for his valuable guidance and encouragement during my writing of the thesis. I was very fortunate to find an advisor who shares my enthusiasm with this topic.

I am also very grateful to professors Nancy Leveson, David Simchi-Levi, George Apostolakis, Thomas Roemer, Daniel Whitney, Daniel Frey of MIT, Paul Carlile of Boston University, as well as professors Marco Iansiti and David Yoffie of Harvard Business School. Their teachings have really taught me about holistic thinking. And many of their concepts are the foundation of the thesis.

I am eager to acknowledge and thank Steve Lipner, Bilal Alam, Jiri Richter, David Ladd, Michael Howard, Andrew Cushman, Jon DeVaan, Paul Oka, Barbara Chung, Philip DesAutels of Microsoft Corporation, as well as Yoav Shapira of Apache Software Foundation for providing valuable data and sharing their insights with me. Without their assistance[1], this thesis will never materialize.

Finally, I would like to thank my parents and my fellowship friends at Boston Chinese Bible Study Group. The thesis would not have been completed without their constant prayers. Furthermore, my utmost gratitude goes to my personal Savior Jesus Christ whose unfailing love has sustained and strengthened me until this very day.

[1] The names and job titles of whom I have interviewed are included in Appendix B.

1. Introduction

1.1 Motivation

Software security has first caught the public's attention in1999 with the Melissa computer virus outbreak, where more than one million[2] PCs in North America alone were estimated to be infected. Now after seven years, our security situations do not look any better. It is true that the computer industry and our government have put in a lot of efforts to secure our cyber workplaces and homes. For example, we now have much more advanced firewalls, intrusion detection systems and antivirus software than in 1999. And enterprises and consumers alike have poured in hundreds of millions of dollars to purchase and install such security devices either at the network perimeters or on the desktops. Nowadays you can hardly buy a new PC without some security software preinstalled. In addition, several anti computer crime legislations have been passed and FBI has successfully prosecuted several high profile cyber criminals. However, all these actions albeit helpful, have failed to address the root cause of the problem – insecure software.

First of all, no matter how good those security devices are, they are all composed of fallible software. Piling them up on top of vulnerable applications not only do not decrease the total vulnerabilities of the entire system, but also introduces new failure modes, which arise from the unexpected interactions between the security software and the application software, as well as human operational errors due to the increased complexity of the systems. One 2005 Yankee report (Jaquith, 2005) has found that there

[2] Details at http://www.usdoj.gov/criminal/cybercrime/melissa.htm

were 77 vulnerabilities found in security products between 2004 and May 2005, far exceeding the ones uncovered in the Microsoft products (which the formers are supposed to protect), during the same period (figure 1). Secondly, because security is hardly considered in our software designs, we have ended up with an IT infrastructure not only full of easily exploitable security defects, but also void of effective monitoring, detecting and reporting functions to the extent that most of the security crimes went undetected (O'Connor, 2006). These make the cybercrime legislations and enforcements crippled at best and toothless at worst.

Since all these external forces do not yield satisfactory result, we need to look inward at the software itself. Hence a question naturally arises - how to make software inherently secure?

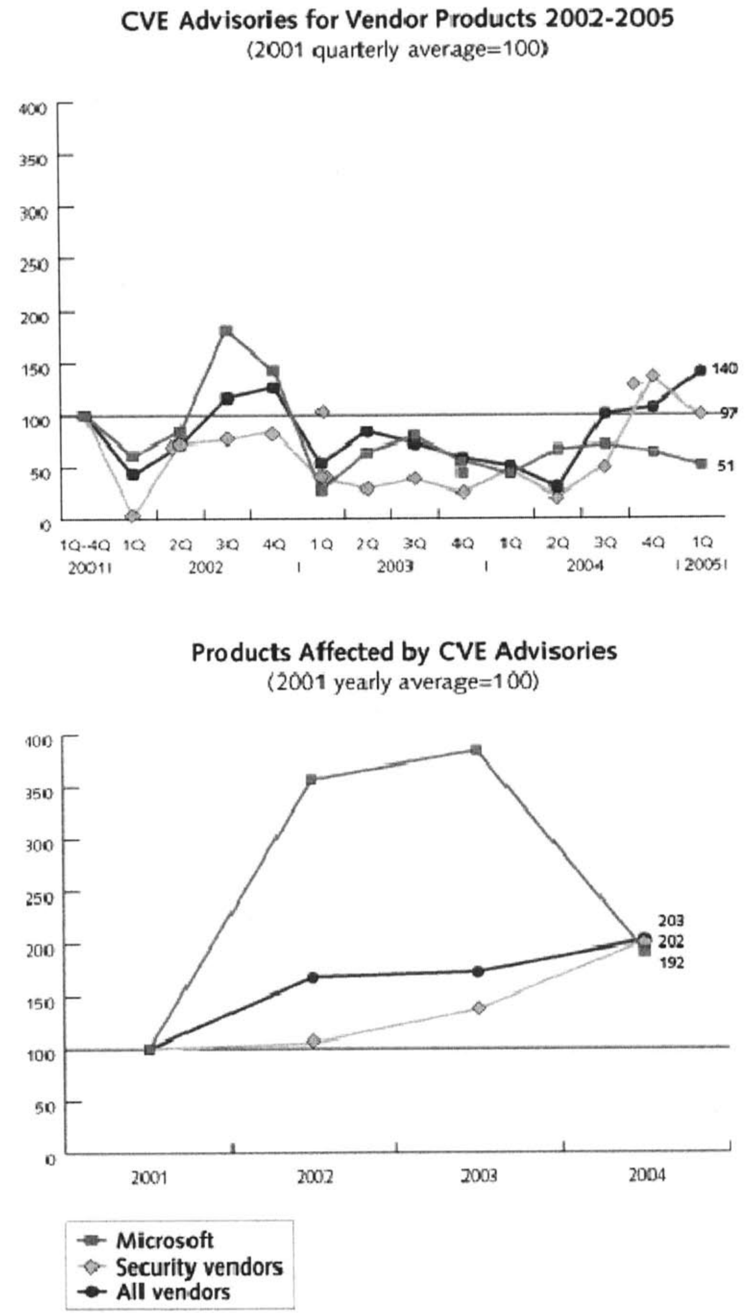


Figure 1 Security Vendor Vulnerabilities Increase *(Source: Jaquith, 2005)*

1.2 Objectives

The main objective of my thesis is to explore how to design security into software. The subject can be further broken down into a sequence of sub-questions:

- What is security within the software context?
- Are there any unique characteristics that software security has to the extent that our existing software design methods seem no longer sufficient?
- What should the software development organizations do in order to produce secure software?
- Since it is unlikely one development method will fit all projects, what are the pros and cons of them and their application domains?

1.3 Structure of the Thesis

The thesis is constructed in the following manner:

- **Chapter 1 "Introduction"** discusses the author's motivation, key objectives, and the structure of the thesis.

- **Chapter 2 "Literature Review"** provides an overview of the software security field, including the definition of software security, its current state and the research having been carried out in this aspect.

- **Chapter 3 "Case Study: Microsoft Internet Information Services (IIS) Web Server"** examines the Security Development Lifecycle (SDL) used in Microsoft IIS product, which is the mostly used web server by Fortune 1000 companies. The SDL, which is the standard process used across Microsoft, includes many suggestions from the literature review in Chapter 2. We will analyze its implementation in real life, reflect on its achievements, and give suggestions on areas of improvement.

- **Chapter 4 "Case Study: Apache HTTP Web Server"** investigates how security is being handled by one of the most popular and successful product of the Open Source Software (OSS) movement – Apache HTTP Web Server, which has dominated the world wide web server market ever since its birth in 1995. As a typical OSS, its development method is at odds with those of the traditional software vendors such as Microsoft. However, it security scorecard is remarkable. Here we examine its development process, identify its strength and weakness in light of the research recommendations from Chapter 2, and make suggestions on areas of improvement.

- **Chapter 5 "Conclusion"** assimilates a framework that help to incorporate security into software design based upon the principles from the research and the actual practices from the industry's leading practitioners (i.e. Microsoft and Apache). This entails recommendations as to how to design security into software development process. Finally we elaborate what the future holds in this field of research.

- **Chapter 6 "Future Work"** discusses other remaining issues beyond the scope of the thesis and identifies some opportunities for further work.

2. Literature Review

2.1 What is security within the software context?

Security is a fuzzy and subjective concept. Different people have different interpretations. Although Merriam-Webster online dictionary has just 4 entries for this word, with "free from danger or free from risk of loss" more or less fit for the paper's purpose, a quick search on the web yields more than two dozen different explanations around this theme.

Security is also a relative term. There is no such thing of 100% secure. Security can only be rationally discussed within a context (e.g. secure under what circumstances, against whom etc.) and usually implies trade off with other desired features such as cost and ease of use (Viega and McGraw, 2001).

For our purpose, software security boils down to design software that protects the confidentiality, integrity, and availability of resources under the control of the authorized user even when some malicious forces try to make it behave differently (Howard and LeBlanc, 2002). This calls for software to be designed

- With the necessary and usable security functions (e.g. user authentication, access authorizations, data protection etc.) in order to provide a safe environment to serve the main purpose of the software, and
- Without security related design flaws (e.g. buffer overflow, least privilege violation etc.)

Not all the software products need to have their own security functions. For example, the Microsoft Paint program does not have its own user authentication or file protection features, precisely because these have already been provided by the operation system. So whether to design security functions into the software is context dependent. However, designing the product without security vulnerabilities is a universal requirement for all software.

2.2 What are the main characteristics of software security?

"So isn't security just another software feature?" Someone may ask. The answer is yes and no. On one hand, security is indeed one of the many attributes of the software. On the other hand, it is different from the rest of the crowd on the following grounds:

- Software security deals with the whole system rather than with individual software modules or components. It is an emergent property of the system, not a component property (Leveson, 1995; Viega and McGraw, 2001). You can not design secure components in isolation and then put them together and hope the whole is secure. Not only because serious security complications usually come from the interfaces and component interactions, but also because designs that make perfect security sense at the component level may even compromise the security of the system.

  To illustrate the latter point, let's look at a real life example. In many financial institution's websites, once a user has typed in the wrong passwords for a consecutive number of times, his/her account will be locked and the user has to call the customer service to reactivate the account. If we consider the standalone authentication module, whose main security function is not to let the bad guy in, this is a reasonable design because it reduces the risk that people can guess out the password by brutal force such as dictionary attack. But viewing from the system's level, things are quite different. As a system, it needs to care about not only the access control, which is the primary concern of the authentication module, but also its availability to the authorized users. Although such design will deny unauthorized access, it will also lock out the good users whose usernames happen to have been tried out (but password mismatched) by the attackers.

Therefore, security must be considered under the whole product context.

- Software security is also about trade offs in system design (Leveson, 1995). Except for the special purpose security software such as firewalls and intrusion detection systems, security is hardly the main purpose that we build the software. It is therefore often found conflicting with other functional requirements or system attributes such as performance and usability, and hence demanding elaborate design compromises and careful risk management (Viega and McGraw, 2001).

- Software security is more than just writing secure code (Leveson, 1995; Zipkin, 2005). It requires well-coordinated team efforts throughout the entire product development organization, including the top management demonstrating the commitment to security and instilling this vision from the top brass down to the foot soldiers, engineering management team setting up the proper team structure, development processes and incentives, marketing getting the correct security requirements, product management team analyzing and synthesizing them into the right security specifications, architects making sound security design decisions and tradeoffs, developers and testers following best security practices in implementation and testing, documentation and product deployment team providing easy to use operational documentations and training to the users, security response team monitoring and promptly reacting to security issues uncovered in the field, maintenance team supplying backward compatible security patches in a timely fashion and adapting the product against the latest unforeseen security threats, and finally release management team having a good end of life or upgrade strategy for the customers without compromising the security (Howard and LeBlanc, 2002; Viega and McGraw, 2001; Bishop, 2003; Frincke and Bishop, 2004).

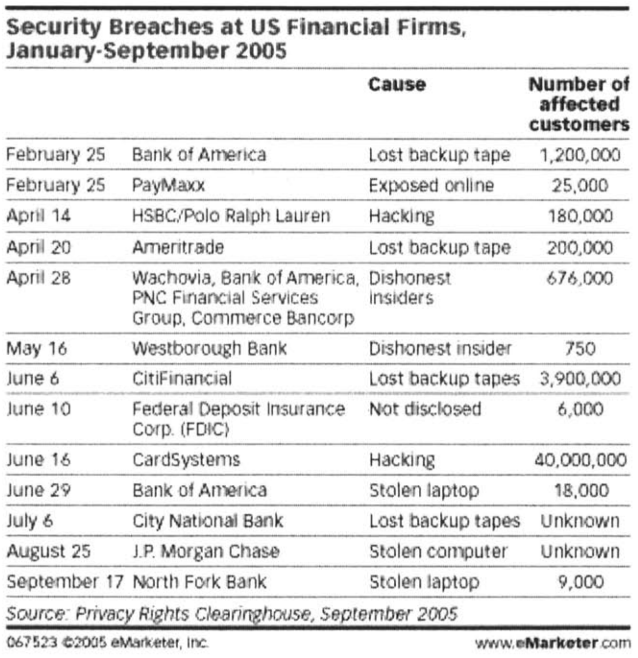
- Software security is often mistaken for reliability. IEEE 610.12-1990 defines reliability as "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." Hence software reliability is about the prevention, detection, and correction of software defects (Rosenberg et al, 1998), which include security related design flaws. However, it does not require the software to have security functions. Furthermore, reliability is usually defined within a normal and benign operating environment but security is specifically designed for a hostile context. Therefore although these two concepts have overlaps, they are not the same.

2.3 Current state of software security

As our society getting increasingly dependent upon computers hence the software that controls them, software security has become a problem that affects everyone in the world even for people has never heard of the computer. This is because so much of the world's infrastructures (e.g. telecommunication, commerce, medical, power, water, transportation systems etc.) are now all dependent upon the software. Anyone reads newspaper or watches TV or listens to the radio will agree that we have serious security problems with the software. But how big is this problem?

Symantec's latest "Internet Security Threat Report" (Symantec, 2006), which covers the six-month period from July 5, 2005, to December 5, 2005, shows us the following disturbing picture:

- In the past the hackers mainly consisted of people motivated by curiosity, or the desire to show off superior technical skill, or pure vandalism. As those people grow older, many of them often get tired of this hobby and find their interests else where. However, the recent data shows that current attacks are increasingly for profit. For example, the financial services sector has quickly become the most frequently attacked industry from the number 3 spot in the first half of 2005 (some of the high profiled security breaches in US financial firms are shown in figure 2). And while past attacks seek to destroy the data, today's attacks are increasingly aimed at stealing the sensitive information without the users ever noticing the intrusions having taken place. A Gartner study in March reported that more and more web server attacks have been used to steal user financial information for identity theft, and Denial of Services (DoS) attacks to blackmail small to medium sized enterprises for payment to avoid or terminate the attack, and furthermore these attacks are increasingly funded by organized crime (Pescatore, 2006). These all suggest the dangerous trend of more sophisticated, diversified, and even organized attacks used for cybercrime (Jaquith, 2005).



Security Breaches at US Financial Firms, January-September 2005

		Cause	Number of affected customers
February 25	Bank of America	Lost backup tape	1,200,000
February 25	PayMaxx	Exposed online	25,000
April 14	HSBC/Polo Ralph Lauren	Hacking	180,000
April 20	Ameritrade	Lost backup tape	200,000
April 28	Wachovia, Bank of America, PNC Financial Services Group, Commerce Bancorp	Dishonest insiders	676,000
May 16	Westborough Bank	Dishonest insider	750
June 6	CitiFinancial	Lost backup tapes	3,900,000
June 10	Federal Deposit Insurance Corp. (FDIC)	Not disclosed	6,000
June 16	CardSystems	Hacking	40,000,000
June 29	Bank of America	Stolen laptop	18,000
July 6	City National Bank	Lost backup tapes	Unknown
August 25	J.P. Morgan Chase	Stolen computer	Unknown
September 17	North Fork Bank	Stolen laptop	9,000

Source: Privacy Rights Clearinghouse, September 2005

067523 ©2005 eMarketer, Inc. www.eMarketer.com

Figure 2 Security Breaches at US Financial Firms, January-September 2005 *(Source: eMarketer)*

- In 2005, there are 3,767 documented vulnerabilities, which is an increase of 40% from the 2,691 in 2004 (figure 3), and is the highest recorded number since Symantec's establishment of the vulnerability database in 1998. The vast majority of them are rated as either moderate or high severity, with only small number of low severity ones. And 79% of them were easily to exploitable, which is an increase of 8% from the same period in 2004. 69% of the vulnerabilities were in Web applications, increasing significantly from 49% in 2004 (figure 4).

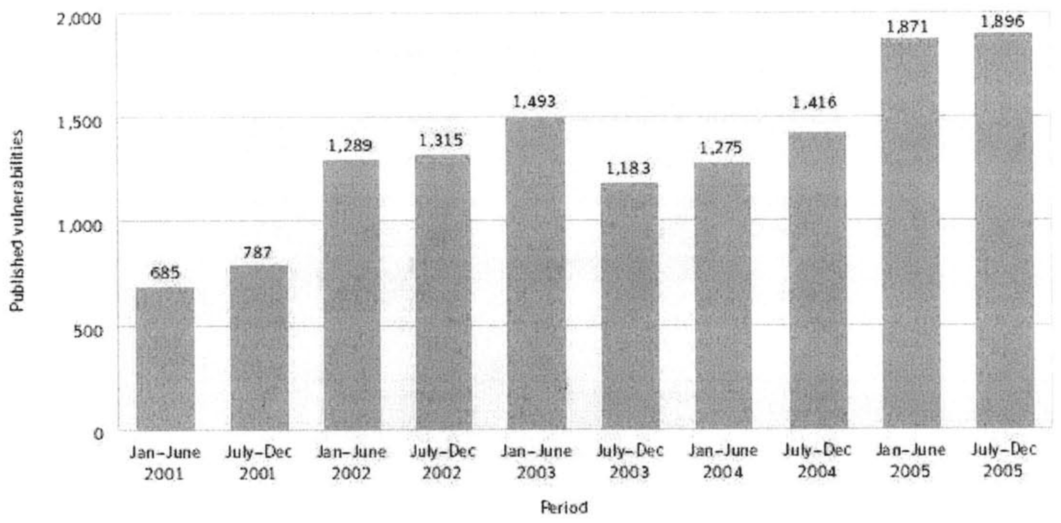


Figure 3 Total volume of vulnerabilities, 2001-2005 *(Source: Symantec Corporation)*

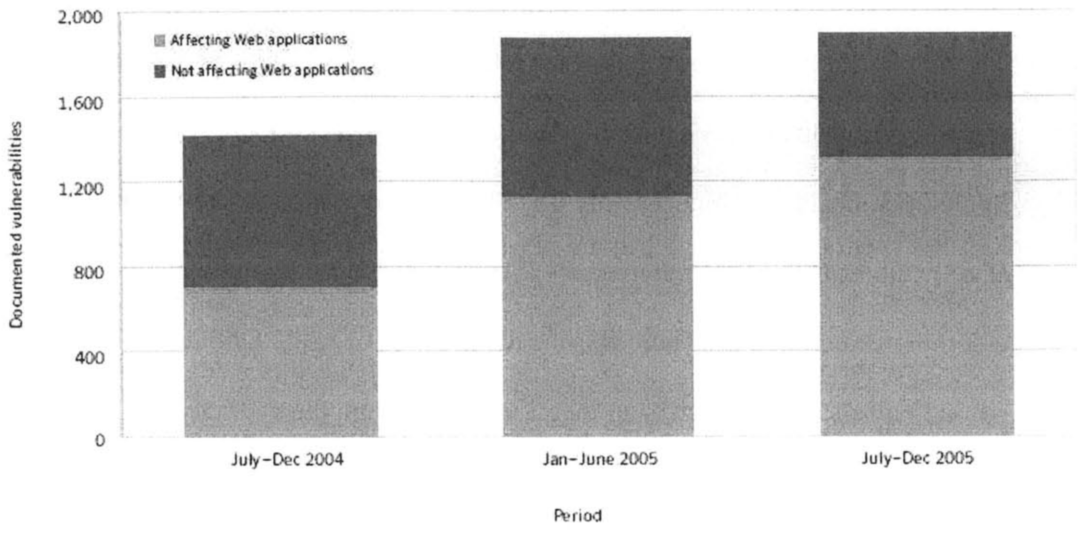


Figure 4 Web application vulnerabilities *(Source: Symantec Corporation)*

- During the second half of 2005, the average time from when the vulnerability is publicly disclosed to the time exploit code being developed was 6.8 days. And the average time that the vendor takes to develop and release a patch to mend the hole (a.k.a. time to patch) was 49 days. Subtracting the first number from the second one gives us an average "window of exposure" time of 42 days, which is the period that the vulnerable application will be open to potential compromise unless the users take other protective measures (often expensive and cumbersome). This is a significant improvement from 64 days in the first half of the year but a slight decline from 40 days in the same period of last year.

- The number of detected bots (short for robot), which are comprised computers remotely controlled by the hackers through installed backdoors[3] , was on average 10,000 per day (figure 5). They are often being used to launch Denial of Services (DoS) attacks against the enterprise's Web site and cause revenue disruption for e-commerce companies. There was a daily average of 1,402 DoS attacks for the second half of 2005, which is an increase of 51% from the average of 927 attacks per day over the first half of the year (figure 6). Symantec believes due to the large number of high-profile vulnerabilities in Web applications, bot networks will grow rapidly.

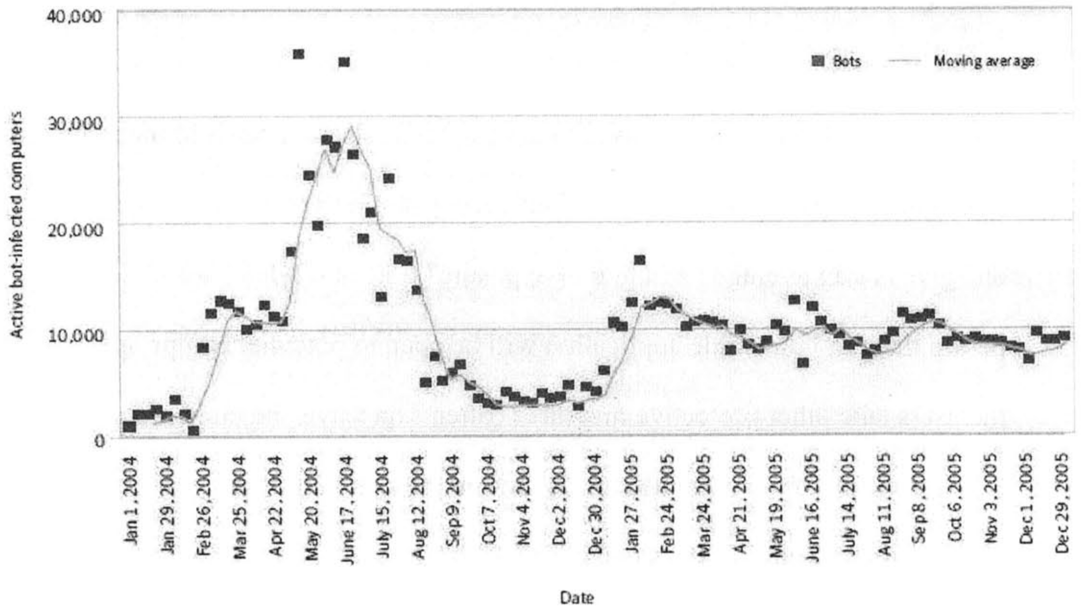


Figure 5 Bot-infected computers detected per day *(Source: Symantec Corporation)*

[3] malicious software that is self-installed without user notice by exploiting vulnerabilities in the victim's computer, and often can be remotely upgraded to take advantage of newly found vulnerabilities

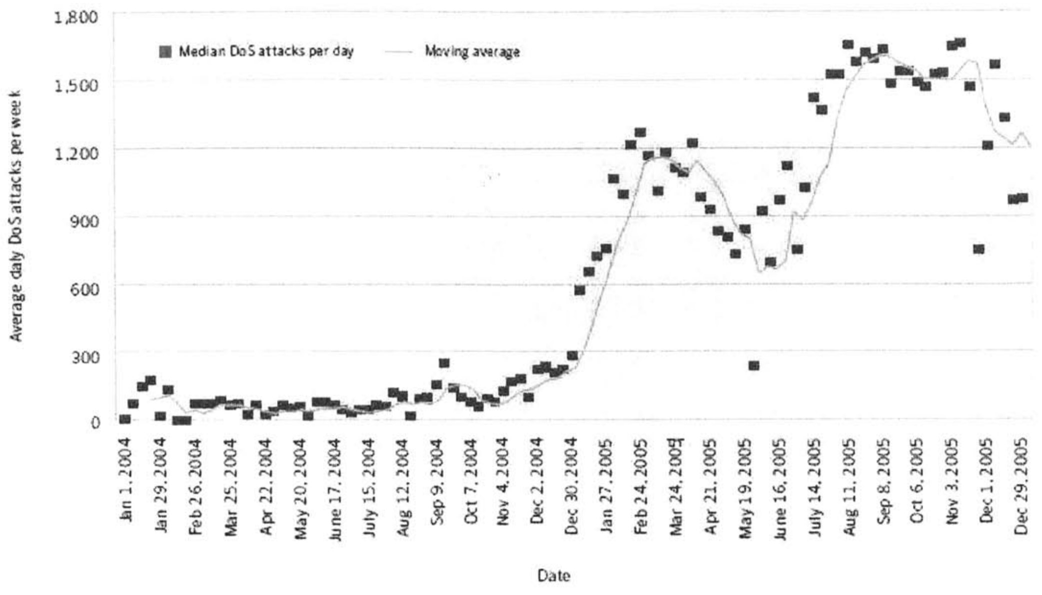


Figure 6 DoS attacks per week *(Source: Symantec Corporation)*

Giving the above statistics, the consumers are scared (figure 7). John Pescatore, Vice President for Internet Security at Gartner, said that he foresees a "flattening" of e-commerce because of the alarming increase of cybercrimes. "In our survey, we found that people are no longer sure about online bill payment. Those who have been hit are less likely to accept new things." (eMarketer, 2004).

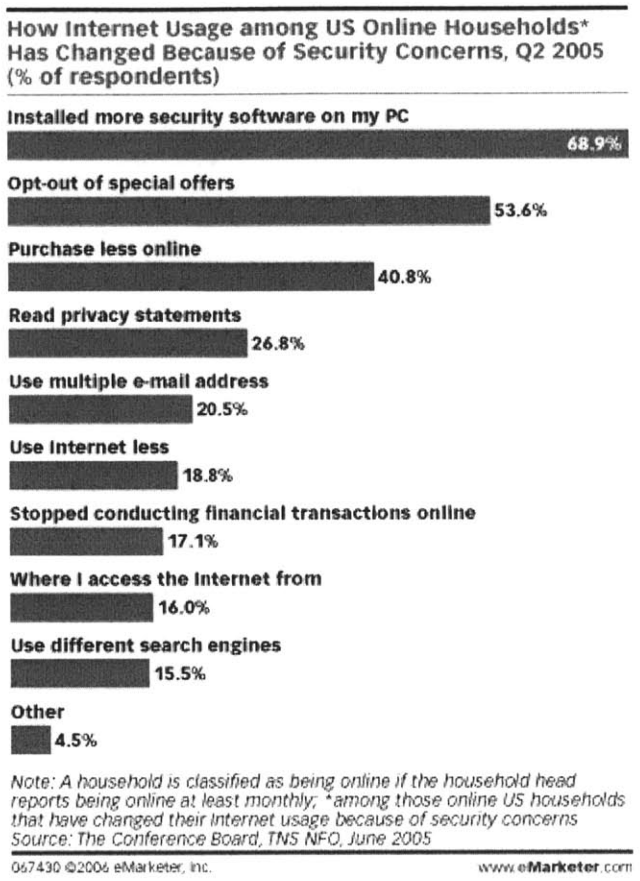


Figure 7 How Internet Usage among US Online Households Has Changed Because of Security Concerns, Q2 2005 *(Source: eMarketer)*

In sum, we now have attackers that are getting more sophisticated and motivated, and with tens of thousands of bots at their disposal to launch attacks, software vulnerabilities that are at record high and on the rise, users that have the prolonged window of exposure for 6 weeks on average, and finally 40% of internet users are so scared as to scale back their online activities. So what have our software industry done in response to such threats?

With the worldwide spending on IT security service being $16 billion this year and projected to reach $24.6 billion by 2009 with a compound annual growth rate (CAGR) of 11.8% (Souza et al, 2005), one would think with such enormous investment and public interest our smart engineers and market economy will soon solve the problem just as they did back in the 1960s to solve the quality issue for the American automobile industry. But unfortunately the response is no due to the following reasons.

First of all, security is a hard problem rooted in the nature of the software itself and likely gets worse as software becomes more complex. From engineering's perspective software is one of its own kinds. It is an intangible resulted from an intellectual thought process with the goal to accomplish certain functions which are physically impossible or infeasible to achieve by machines. There are no physical laws or limitations to constraint or validate its design space and the designer usually don't really know how the end result looks like until it has been produced. This makes software design and validation inherently challenging for nonfunctional hence elusive requirements such as security. In addition, because software can be changed without the obvious penalty such as retooling or manufacturing as it is for the hardware, such flexibility is often found to be its curse - "Software is the resting place of afterthoughts" (Leveson, 2004), and more than often resulting in a pile of patchwork. Therefore software design is dominated by "good enough" mentality with security rarely being considered until incidents occur in the field. Furthermore, software vendors are pressed to modify their design constantly to meet changing user needs while trying to build complex new products and technologies. Hence even an initially secure product risk of having its quality degraded over the time. As the

Internet has connected computers everywhere, it adds another magnitude of complexity into software design (Cusumano and Yoffie, 1998). As the consequence, complexity has inevitably increased in the software and will likely continue to do so in the future. However, "complexity is the enemy of security" (Wagner, 2006) - complex software is not only difficult to design and secure right from the beginning, but also hard to understand and correct afterwards (Viega and McGraw, 2001).

Secondly, we are not investing in the right solution. Although people are paying more attention to software security by increasing their investments in specialty security solutions such as VPN, firewall, intrusion detection and prevention system, and anti-virus software (eMarketer, 2005), establishing IT security management process to screen products for vulnerability, improving corporate IT infrastructure via risk assessment and mitigation measures, and taking into account of threats originated from both internal and external (Burke et al, 2005), they are only looking at the security issue in a piecemeal fashion. And they are not addressing the root cause of the problem, which is the bad software. Furthermore, Many of the solutions are themselves software products and hence susceptible to the same type of vulnerabilities in the software that they try to protect. And indeed there have already been reports about critical code execution vulnerabilities found in several major antivirus products (Naraine, 2006).

Thirdly, the common practice of handling software security issues, which is to release the product with little security considerations and then rush out a security patch when a vulnerability becomes publicized or publicly exploited, is hugely ineffective (Viega and

McGraw, 2001). The main problems to this "penetrate-and-patch" approach (sometimes can be more appropriately called as "patch-and-pray") are the following:

- The total time taken by the vendor to come up with a fix, making it available to all its customers, and have them all patched up far exceeds the time that the hackers need to create an exploit and make it available over the Internet for the entire world to use. For example, there have been experiments where an out-of-the-box Windows XP computer (without any service pack) is connected to the Internet. Within an average of 15 minutes, which is significantly shorter than the amount of time one would need to download and install the security pack from Microsoft, it become infected (Wagner, 2006). Furthermore, the vendors can only fix reported vulnerabilities while the attacker can discover and make use of unknown problems without the vendor ever noticing them (Viega and McGraw, 2001).
- Even when the security patch is available before the attackers launch an attack, there are situations where the security patch not being applied either due to the owners' negligence as evident by the tens of thousands of bots, or its incompatibility with the other key applications that the users are dependent upon.
- Furthermore, above certain level of code complexity, fixing a known vulnerability is not only likely to introduce a new, unknown, and possibly worse security hole (Geer et al, 2003) but also prohibitively expensive than if designing it out before the release of the product (Brooks, 1995).

2.4 *How to design security into software*

Many security experts have pointed out that we can only achieve secure software by incorporating security considerations throughout the Software Development Life Cycle (SDLC) (NCSP, 2004). One such framework is shown below (figure 7). And we have synthesized their views and present them in detail in the following subsections.

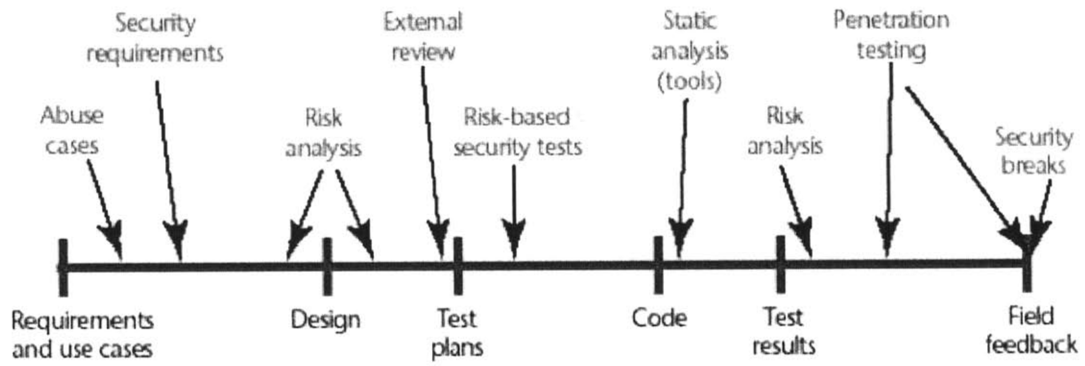


Figure 8 "Software security best practices applied to various software artifacts. Although the artifacts are laid out according to a traditional waterfall model in this picture, most organizations follow an iterative approach today, which means that best practices will be cycled through more than once as the software evolves" *(Source: McGraw, 2004)*

2.4.1 Requirements and Specifications

Security requirements should be separated from the other requirement so that they can be reviewed, designed and tested explicitly in later stages of the SDLC (SI, 2005).Good security requirements should consist of the following aspects:

- Functional objectives (i.e. what needs to be protected, from whom, and for how long) (McGraw, 2004),
- The undesired behaviors that needs to be prevented (i.e. what it should not do) (Williams and MacDonald, 2006)
- The desired emergent systems properties (e.g. how the system should behave under different attack scenarios) (NSCP, 2004).

- Depending on the business environment the product will be operating in, the necessary standards and regulatory policies should also be included in the requirements (SI, 2005).

Once these requirements are in place, all stakeholders, such as representative from the marketing department (i.e. any functional groups that intimately understand the customer environment and its business operation), internal security expert, engineering lead, and quality assurance representative etc., should team together to analyze and translate these information into the proper specifications. Since security is always relative and within a context, both what (i.e. what needs to be done) and why (i.e. why we have to do that) should be recorded in the specifications and in an easy to understand manner (Viega and McGraw, 2001).

Methods such as abuse case[4] (McGraw, 2004), threat model[5] (Howard, 2003), attack pattern[6] (NCSP, 2004) are often suggested to be constructed as early on as this phase for security requirement analysis and then used onwards in the subsequent design, implementation and test phases.

2.4.2 Architecture and Design

Having generated the correct security specifications, which are half of the battle, we need to translate them into secure product architecture and design. Viega and McGraw (2001) strongly recommend a risk management approach be taken at this stage. The idea is to

[4] Similar to the use case which describes how a legitimate user normally uses the system, the abuse case depicts the situation a hacker tries to penetrate the system.
[5] See Appendix A
[6] See Appendix A

first rank these security risks identified in the specification phase in the order of their severity (e.g. business consequences to the customer, loss of market share or reputation to the company etc.) and other criterion such as their probabilities of occurring. Then come up with the possible solutions to either eliminate or mitigate these risks. Based on the ranking of the risks and the costs of their solutions, we then conduct a cost benefit analysis to further refine the security specifications and clarify the appropriate solution space. This information can then be fed into the architecture and design team, where further design tradeoff may occur among security features and other project goals such as usability, efficiency etc. Such compromise should always be based on the business ground.

When designing the overall structure of the system, the resulting system decomposition must be evaluated against the security requirements. Additional scrutiny must be applied to the modules that provide the security functions or access to the protected data (SI, 2005).

Besides risk management, secure system must be architected and designed upon common security principles. One such guideline that was proposed by Saltzer and Schroeder (1975) and recommended by National Cyber Security Partnership (NCSP, 2004) is repeated verbatim as the following:

1. Keep it Simple: Keep the design as simple as possible. Complexity is the enemy to security (Viega and McGraw, 2001).

2. Fail-safe defaults: Base access decisions on permission rather than exclusion.
3. Complete mediation: Every access to every object must be checked for authority.
4. Open design: The design should not be secret.
5. Separation of privilege: Where feasible, separate a privileged process into one process doing the special work requiring high privilege while the other(s) taking on the mundane work with regular privilege.
6. Least privilege: Every program and every user of the system should operate using the least set of privileges necessary to complete the job.
7. Least common mechanism: Minimize the amount of mechanism common to more than one user and depended on by all users.
8. Psychological acceptability: It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.

Since we can never produce 100% secure software, instead of just incorporating risk elimination and mitigation into the design and hoping for the best, architects need to consider whether to include auditing, tracing, and monitoring capabilities (e.g. to know who did what, at when, and how) into the system design (Viega and McGraw, 2001) for in-accident signaling as well as post-accident forensic analysis.

Human is the weakest link. Matt Bishop (1995) claimed more than 90% of security failures are caused by user errors. Therefore no matter how many security functions we jam into the product, if they are not usable in the eyes of the customers, they are useless.

However, researches have found the standard UI design can not be readily applied to security. New security usability guidelines such as the following (included below in verbatim) from Whitten and Tygar (1999) are therefore recommended:

- Inform the user about the security related task that they need to perform;
- Help the users to follow the procedure to carry out these tasks;
- Design the procedure so that misuse of the steps will not cause dangerous errors;
- Design the procedure so that people feel comfortable about continuing to use them;

2.4.3 Implementation

Software can never be secure if it is not coded correctly. Secure implementation requires:

- Educate developers to understand the threats posed by common insecure coding practices such as using strcpy() without validating input data length against the allocated buffer size, and teach them the alternative safe methods (Viega and McGraw, 2001);
- Perform peer review before code being allowed to be submitted to catch implementation errors as early as possible (Viega and McGraw, 2001). Some recommends having separate security review in addition to the normal code review (SI, 2005);
- Have security experts audit high-risk areas of the code (Williams and MacDonald, 2006);

- Choose the proper programming language for the project. C and C++ are efficient and have large developer communities; however, they have abusable properties such as pointers and lack of strong typing, which can easily lead to security vulnerabilities than other languages such as Java and Ada (NCSP, 2004). Team must take the language's security implication into account.
- Use static code analyzers and scanners to automatically detect common vulnerabilities in the code (NCSP, 2004).
- Mandate unit testing for all security related modules as well as code that have access to protected data or directly interact with the environment (user, network, file system etc.), and error conditions that can not be reached under system testing (SI, 2005). Ideally the developers should write unit test cases for each other and submit them into a unit test library (SI, 2005).
- Make security testing part of the nightly build process. It should include both positive and negative tests, and be automated (Jaquith, 2005).

Many experts also recommend setting up and enforce secure coding standard to guide developers on how to write secure code (Williams and MacDonald, 2006). A common developer guideline for this purpose is shown below in verbatim (NCSP, 2004):

1. Validate Input and Output
2. Fail Securely (Closed)
3. Keep it Simple
4. Use and Reuse Trusted Components

5. Defense in Depth
6. Security By Obscurity Won't Work
7. Least Privilege: provide only the privileges absolutely required
8. Compartmentalization (Separation of Privileges)
9. No homegrown encryption algorithms
10. Encryption of all communication must be possible
11. No transmission of passwords in plain text
12. Secure default configuration
13. Secure delivery
14. No back doors

2.4.4 Testing

Testing is the final stage that we still have the chance to recover vulnerabilities before the product being released to the public. Although the traditional functional and system level testing should still be used to examine if the system behaves according to the security specifications, true security testing requires a lot more than that.

Good security specifications elaborate what the system should do and should not do. It is not difficult to write automated test cases to evaluate against these clearly defined objectives. However, it is so often what the system does in between create the security vulnerability (figure 8). Thompson (2003) used a famous vulnerability in Windows NT RDISK utility as an example to illustrate such issues: RDISK is used by system administrators to create an Emergency Repair Disk for the PC. As one of its intended functions, it creates a backup file of the important Windows Registry with the right access permission. However, a temporary copy of the registry is also created during the process, but with universal read permission that allows access to the potential attackers. Such subtle security hole is unlikely to be noticed by automated tests and can only be detected by highly skilled testers (Thompson, 2003).

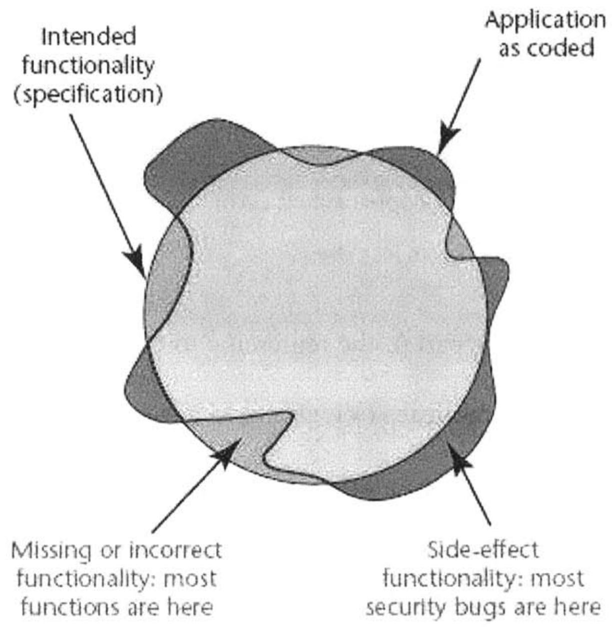

Intended functionality (specification)
Application as coded
Missing or incorrect functionality: most functions are here
Side-effect functionality: most security bugs are here

Figure 9 "Intended versus implemented software behavior in applications. The circle represents the software's intended behavior as defined by the specification. The amorphous shape superimposed on it represents the application's true behavior. Most security bugs lay in the areas of the figure beyond the circle, as side effects of normal application functionality." *(Source: Thompson, 2003)*

Some recommended security testing strategies are:

- Design test cases based on the risks identified in the requirement gathering and design phases (i.e. attack patterns, threat models, abuse cases etc.) (NSCP, 2004). Howard and LeBlanc (2002) suggest special attention be paid to the interfaces among the modules and inject faulty data into them as negative testing .
- Think creatively about the ways that hackers can attack and design test cases (i.e. penetration tests) accordingly (Viega and McGraw, 2001). Having a diverse group of testers will help in this regard (Thompson, 2003).

- If the product needs other applications in order to provide a complete solution for the customer or if it is often used in combination with other applications, then we need to test them together because there could be security implications from their interactions (Howard and LeBlanc, 2002).
- Incorporate automated security tests, which scan for the common vulnerabilities (e.g. buffer overflow, memory leak etc.), into the daily build process and automatically generate bug report when error is found (Williams and MacDonald, 2006);
- Include security tests as part of the regression to ensure new code does not reintroduce past vulnerabilities (Viega and McGraw, 2001);
- Execute code coverage tests and review uncovered code for security vulnerabilities (e.g. Trojan horse etc.) (Viega and McGraw, 2001);
- Reflect on the vulnerabilities slipped through testing and subsequently uncovered in the field and improve testing techniques (Thompson, 2003).
- Develop secure testing guidelines or checklist. Thompson's paper gave a sample guideline as the following (Thompson, 2003):
    - Test for dependency insecurities and failures (e.g. examine how the system behaves if one of its external library files doesn't get loaded);
    - Examine for design insecurities (e.g. check for test APIs or debug harness added by the designer but forgot to remove. These could leave a security backdoor for hacker to use);
    - Inspect for implementation insecurities (e.g. verify public APIs that have properly validated their input parameters)

Any security defects uncovered at this stage should be channeled back to the original design and development team not only for bug fix but also for review to see if improvement in process or practice is needed.

2.4.5 Documentation

Every threat that has been mitigated as opposed to be eliminated as well as security concerns should be documented. They should be explained in the appropriate place in the product documentation with the level of details that are useful to the person who cares about such risk (Howard and LeBlanc, 2002).

Howard and LeBlanc suggest the documentation should have a "Security Best Practices" section and provide the readers with step-by-step guide to use the product securely. It should focus on helping the user to get things done and avoid technical details that are interesting to some but unnecessary to complete the task. Such additional information can be included in the references or a bibliography section (Howard and LeBlanc, 2002).

2.4.6 Post-Deployment Support

Sooner or later security bugs will be reported in the field demanding fast resolution. It is imperative that the development organization have a responsive process and good configuration management practice (figure 9) in place to be able to resolve critical vulnerabilities in standalone patches as opposed to upgrade the entire product, while minimizing the impact to other development efforts (Williams and MacDonald, 2006).

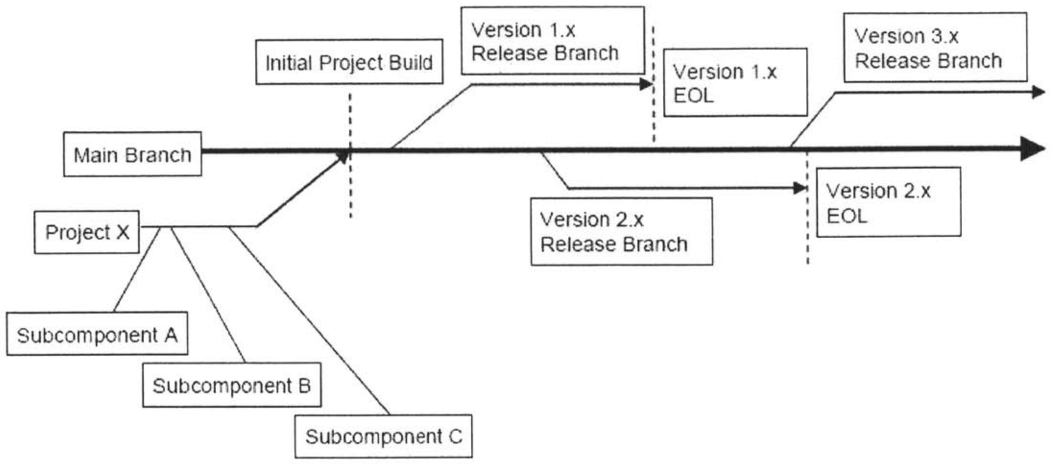


Figure 10 Software Configuration Management Best Practices *(Source: Williams and MacDonald, 2006)*

The security fix should be thoroughly verified that not only the root cause has been addressed as opposed to the symptom, but also no new flaw is introduced and no existing functionality is broken (NSCP, 2004). If this vulnerability is critical, the development team should also inspect other related products for similar errors (Williams and MacDonald, 2006). Any security defects uncovered at this stage should be investigated to see any improvements in SDLC process or practice is needed.

2.4.7 Management

National Cyber Security Taskforce for Corporate Governance states people, process, and technology as the three key elements for a secure system. None of them are possible without the clear commitment and leadership from the management. Therefore, the NCSP report made the following recommendations (NSCP, 2004):

- Institutionalize the SDLC and keep improving it;

- Peter Drucker said "You can't manage what you can't measure." Therefore set up specific and measurable goal for developing secure software such as reducing the reported vulnerabilities by 50% in the 12-month period after the product is released;
- Create organizational and/or project level positions and staff them with security experts that also have deep understanding of the product. The person/team are the center of excellence in security related matters. They should participate in each stage of the SDLC, help the product team to evaluate and resolve security related issues, and provide security reviews (Viega and McGraw, 2001);
- Provide ongoing security training to the entire product organization (Howard and LeBlanc, 2002);
- Provide the funding to equip the development organization with the necessary tools;

2.4.8 Tools

Designing for security is a vast task. The necessary tools are indispensable for both aiding their work and avoiding human error:

- Requirement mapping tools, where the security specifications can be traced all the way from requirement, system architect, development, testing, to the maintenance phase to ensure they are being properly understood and implemented throughout the SDLC (Williams and MacDonald, 2006);
- Static analyzer and code scanner to automatically detect common vulnerabilities in the code (NSCP, 2004);

- Many security issues arise from the unexpected interaction between the system and the environment. Therefore the testers need to have tools that can simulate such interaction by dynamically injecting data into the tested application and capture the results (Thompson, 2003).

Having reviewed the main theory and research recommendations in the field, let us now examine how security is being tackled in two leading software producers: Microsoft and Apache Foundation.

Although many people are quite negative about the security of Microsoft products, the company has made significant progress in the recent years and spearheaded many security innovations in the commercial software industry. In the following chapter we will look at the security practices used in the development of Microsoft IIS 6.0 product, which is the mostly adopted web server among US Fortune 1000 companies.

Open Source Software use very little traditional software engineering processes but has brought to the world many quality products. We will then examine in chapter 4 how security is being handled in one of its most successful products, the Apache HTTP Web Server, which has dominated the worldwide web server market since 1995.

3. Case Study: Microsoft Internet Information Services (IIS) Web Server[7]

3.1 Brief history of software security at Microsoft

Back in the early days of Microsoft, security wasn't on the company's radar screen. After all, MS-DOS, Microsoft Windows 3.1 and 95 were all designed for single-user environment. Anyone that got onto the PC was automatically deemed trustworthy hence had the full control over the computing resources. Therefore security was really about controlling the physical access to the machine and had little to do with the software itself. It was not until Microsoft entered into the server market (with Windows NT 3.1), whose multi-user environment required the operating system to protect one user/process from the other, that the company began to design security functions into the products.

As the Internet took off which made computers increasingly connected with each other, and more users adopted Microsoft products, the company started to face a series of high-profile security incidents caused by vulnerabilities in its code. For quite some time those security defects were directly dealt by the individual product teams who followed the "penetrate and patch" method. As the company began to realize that such ad hoc approach was suboptimal, a Security Response Team (which eventually evolved to today's Microsoft Security Response Center (MSRC)) was formed in mid-1998 to centralize the handling of all security related incidents and act as the sole interface for the security researchers and customers. In the meantime, an internal Security Task Force was also established to investigate the root causes of the vulnerabilities across the product

[7] The chapter is based on private communications with Microsoft personnel unless otherwise noted.

lines and came up with a set of long term solutions, which included recommendations such as providing security training, mandating code review and security testing, better product development and security response processes etc.

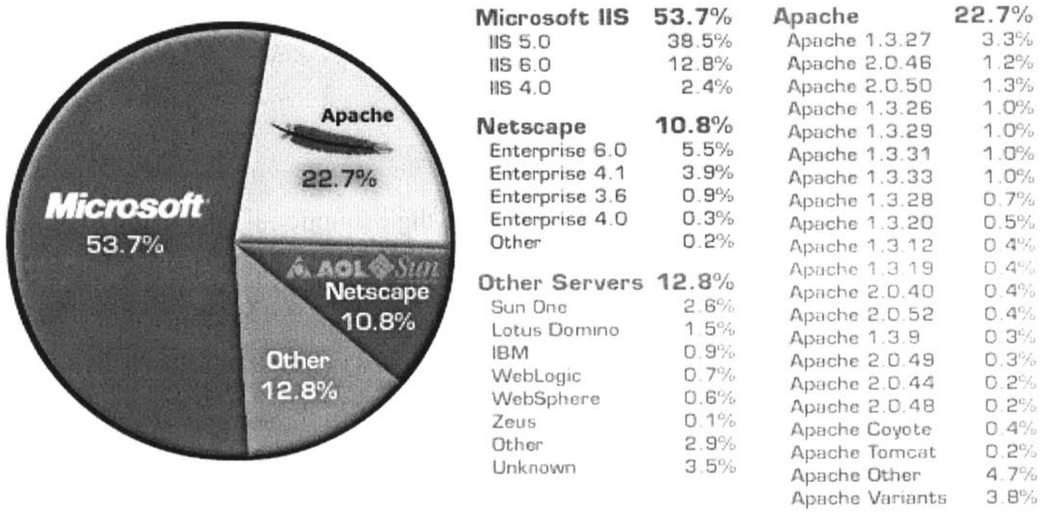
Based on the team's suggestions, Microsoft's Windows Division created a dedicated security program management team - the Secure Windows Initiative (SWI) team, which was chartered to help the product teams to improve the security of their products by providing them with security education, vulnerability detection tools, and development process improvement. A penetration test team, which later became part of the SWI organization, was also established to walk through the Windows 2000 code base to uncover any potential security issues.

Starting with the .NET Framework Common Language Runtime, Microsoft carried out a series of security initiatives and security pushes in the development of several high profile Windows products including Windows Server 2003. Their successes encouraged the company to formally establish Security Development Lifecycle (SDL) in early 2004 as a corporate mandated component of software development process. Each product team can still choose its own development process and method, but has to incorporate the steps that constitute the SDL.

In the following sections we will first examine in detail how security had been tackled in the IIS 6.0 product, which is a component in Windows Server 2003. Then we will introduce the current SDL which includes some interesting improvements over the original IIS approaches.

3.2 *IIS*

Internet Information Services (IIS) is the Web server component of Windows Server operating system. According to Port80 Software, it has dominated the Fortune 1000 enterprises (figure 11).



Microsoft IIS	**53.7%**
IIS 5.0	38.5%
IIS 6.0	12.8%
IIS 4.0	2.4%
Netscape	**10.8%**
Enterprise 6.0	5.5%
Enterprise 4.1	3.9%
Enterprise 3.6	0.9%
Enterprise 4.0	0.3%
Other	0.2%
Other Servers	**12.8%**
Sun One	2.6%
Lotus Domino	1.5%
IBM	0.9%
WebLogic	0.7%
WebSphere	0.6%
Zeus	0.1%
Other	2.9%
Unknown	3.5%

Apache	**22.7%**
Apache 1.3.27	3.3%
Apache 2.0.46	1.2%
Apache 2.0.50	1.3%
Apache 1.3.26	1.0%
Apache 1.3.29	1.0%
Apache 1.3.31	1.0%
Apache 1.3.33	1.0%
Apache 1.3.28	0.7%
Apache 1.3.20	0.5%
Apache 1.3.12	0.4%
Apache 1.3.19	0.4%
Apache 2.0.40	0.4%
Apache 2.0.52	0.4%
Apache 1.3.9	0.3%
Apache 2.0.49	0.3%
Apache 2.0.44	0.2%
Apache 2.0.48	0.2%
Apache Coyote	0.4%
Apache Tomcat	0.2%
Apache Other	4.7%
Apache Variants	3.8%

Figure 11 Web Servers used in Fortune 1000 companies, May 2005 *(Source: Port80, 2005)*

Earlier versions (i.e. pre-6.0) of IIS had many vulnerabilities and were hit particularly hard by the Code Red Worm. However, the latest version of IIS, 6.0, which is an almost complete rewrite (Brown, 2003), has only two[8] vulnerabilities reported since 2003.

3.2.1 Organizational structure

The IIS 6.0 product team consisted of about fifteen software developers, eight program managers (PM), and thirty testers. The PMs authored the technical specifications for the

[8] Details at http://secunia.com/product/1438/

developers, drove the product vision and overall design, and managed the projects. The team as a whole took the responsibility for the security of the product (both current 6.0 and the previous versions).

The team also leveraged the Secure Windows Initiative (SWI) group as a security resource. Two SWI PMs delivered security training to the team and performed design reviews and Final Security Review (FSR) to ensure security issues having been properly addressed and the product was secure enough for shipment.

3.2.2 Specifications

Although the customers are usually good at expressing their functional needs, the team found they often could not articulate their security requirements. Therefore the PMs worked with the developers to come up with their own security specifications, which in most cases were based on the lessons learnt from the past security breaches and sense making.

For example, one security requirement for IIS 6.0 was to have a minimum attack surface. It came from the lesson learnt from the infamous Code Red Worm: IIS used to automatically load all the ISAPI extensions, regardless of whether they were used or not. However, in the Code Red Worm case, one of the unused extensions had a buffer overrun problem and because it was loaded with the server, the server was compromised.

The change in the execution privilege of the server was also the result of the post-mortem analysis of past accidents – the server as a whole is now no longer executed under the all powerful System account but has most of its work done with the less privileged Network Service account. Therefore the impact of potential security breach was limited.

3.2.3 Design

During the design phase, the team used threat modeling to identify the security threats. It was usually performed as part of feature discussion where area owners and anyone interested would get together to brainstorm the possible attacks. They followed Michael Howard's original[9] threat modeling steps to gain better security sense of the system and to identify the components and interfaces vulnerable to attacks (Howard and LeBlanc, 2002).

Security design guidelines were used in the architecture development. Some of key principles are the following:

- Least and separation of privileges. In IIS 6.0, the server work was repartitioned and delegated to two classes of processes: ones require the kernel privilege while the others work under the user mode. Only code requiring high execution privileges were included in the former while the rest were contained in the worker processes, which executed under Network Service account with extremely limited operating system rights. Therefore the compromise of the worker processes would not give the attackers the access to the privileged kernel resources. In addition the

[9] The threat modeling steps were described in the 1st edition of "Writing Secure Code". IIS 6.0 was the first product to have a comprehensive threat model built for the product

work assigned to the high privileged processes was carefully minimized to reduce attack surface[10];

- Defense in depth. It entails using different defense mechanisms and applying them at multiple layers along the attack path. Thus the attackers need to penetrate all the levels in order to breach the application and gain access to the protected data. As an example, the HTTP.sys process was not only hardened against potential buffer overflow, but also equipped with user configurable URL parsing filter to fend off malicious input. In addition, the process logged the HTTP request into a file on the disk before transferring it to the worker process. Therefore even if the worker process died after the handover, the customer would still have a record for forensic analysis (Belani and Muckin, 2004);
- Minimization of attack surface. For example, the IIS was not installed as part of the default Windows 2003 setup. And even after installation, it served only static HTML pages, the more powerful but also potentially more dangerous dynamic HTML pages were not allowed;
- Secure by Default. The team created a wizard to facilitate the secure setup of other related components and chose the default settings that were secure and met the needs of most customers. Therefore customers do not need to change its default configuration to make it more secure;

3.2.4 Implementation

[10] The attack surface of an application is "the union of code, interfaces, services, protocols, and practices available to all users, with a strong focus on what is accessible to unauthenticated users" (Howard, 2004).

In order to assure the secure design gets properly translated into secure code, the team took the following approach:

- Implemented its own secure string manipulation class, which was then used as a shared library across the team to avoid the incorrect URL parsing scattered all over the code. Potentially harmful functions such as strcpy were banned and replaced with safer wrapper functions;
- Mandated coding standard. For example, code readability is the number one requirement for any code submission;
- Used automatic static code analysis tools such as PREfast and PREfix to scan for common vulnerabilities, including some buffer overruns, in the code changes prior to any code submission;
- Conducted code review with special attention paid to the areas of code that are deemed security critical during threat modeling. Code with security ramifications usually required two to three people to do a sit down multi-hour review in order to weed out any security defects;

3.2.5 Testing

The team had 1.5 dedicated testers for security testing. One maintained a set of security regression test suite to detect the re-introduction of known vulnerabilities found in previous versions of the product. The other one worked part-time doing penetration testing in addition to the normal functional testing. Since there was no security testing guideline available at that time, the tester based his penetration test cases on the attack patterns of past incidents. He primarily used targeted fuzz testing[11], which was to inject mutated inputs at different data entry points in the running code. For example, the server was fed with URLs of all kinds of letter and number combinations, and with different HTTP request headers and sizes. All were to ensure the server neither to crash nor to grant access to the attackers. Vulnerabilities were filed with higher priority than functional bugs and had its own security category. They also examined if the problems also occurred in previous versions of IIS, which would then be fixed and included in the next service packs.

In addition to the product team, the penetration team within the SWI group also did its own security auditing. They reviewed the security critical segment of the code and used their own experience and past vulnerability reports as the guide to discover new weakness in the product. At that time their focus was on the entire Windows Server 2003 code base as opposed to its IIS component.

3.2.6 Release

While the IIS 6.0 was in early beta testing stage, Microsoft launched a corporation wide security push. Basically it froze the development work, retrained the entire Windows division for secure programming, and had the entire division exclusively working on detecting and resolving security defects in the code. As the result, IIS PMs, developers and testers took additional security training including the best secure design and testing practices from SWI and helped to train other groups. They then spent two months reviewing and testing their code specifically for security defects. Any uncovered security

[11] Fuzz testing a testing technique which feeds random input to an application. For the details, please see http://en.wikipedia.org/wiki/Fuzz_testing.

vulnerabilities that were remotely exploitable or locally exploitable for elevation of privilege were deemed as "ship-stopper" issues.

Furthermore, the company also hired external consultants after the security push to further scrutinize the entire Windows Server 2003 product including IIS. This consisted of additional code auditing and testing, as well as design review and data flow analysis to ensure secure architecture design. A new compiler that instrumented the code to guard against certain types of stack based buffer overruns at run-time was also introduced and applied by the entire Windows division. The product was also deployed and live tested on Microsoft.com from Beta2 with all 50 servers running IIS 6.0.

Having involved in the product team's development process including the participation in key design and code review meetings, inspection of the design and test documents, evaluation of the internally identified security bugs, the security PMs from SWI monitored and assessed the overall security quality of the product and then advised the product team as well as the upper management on whether the product was security ready for the final release.

3.2.7 Maintenance

Microsoft Security Response Center (MSRC) as the sole customer facing interface for security incidents plays the main role in monitoring the security landscape on behalf of all Microsoft product teams including IIS. Through its relationships with partners, security researchers and product vendors, as well as active monitoring of the hacker discussion lists, MSRC takes a proactive stance in discovering any potential security issues and is usually the first responder to any security incidents (Charles et al, 2006).

MSRC defines the following security response process which is followed by all the product teams: once MSRC is notified of a possible security bug, both the SWI team and the affected product team are engaged. The two teams then work together to weed out false alarm and assess the product impact if it is a real security bug. The development team provides the bug fix, which is regression tested and bundled with other fixes and published according to a monthly release cycle. Then a security bulletin is posted to www.microsoft.com/technet/security to inform the customers and the general public. After the fix becomes available, the SWI team does its own root cause analysis of the incident. They focus on the process improvements that may be needed in order to prevent the problem from reoccurring. That learning is then fed back into the security education process.

SWI also regularly monitors the outside security landscape including vulnerability reports of other vendors' products and notifies teams within Microsoft if any similar issues may exist in their products.

3.3 Evolution to SDL

The security successes of IIS 6.0 and several subsequent products such as SQL Server 2000 Service Pack 3 and Exchange 2000 Server Service Pack 3, which applied many of the security practices used in IIS, eventually convinced Microsoft to establish the

Security Development Lifecycle (SDL) in early 2004 as a corporate mandated software development process (figure 12) for all Microsoft software products that:

- Are intended for non-consumer market, or
- Handle personal or sensitive information, or
- Require Internet connection to perform its intended functions.

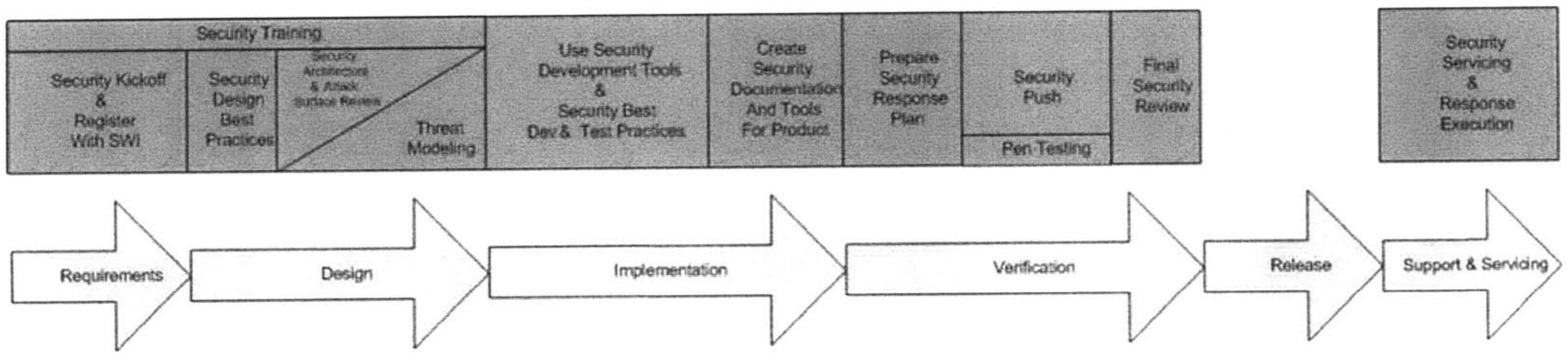

Security Training
Security Kickoff & Register With SWI
Security Design Best Practices
Security Architecture & Attack Surface Review
Threat Modeling
Use Security Development Tools & Security Best Dev & Test Practices
Create Security Documentation And Tools For Product
Prepare Security Response Plan
Security Push
Pen-Testing
Final Security Review
Security Servicing & Response Execution
Requirements
Design
Implementation
Verification
Release
Support & Servicing

Figure 12 Microsoft SDL *(Source: Lipner and Howard, 2005)*

This new paradigm is based on the following principles (Lipner and Howard, 2005):

- Secure by Design: software should be designed inherently secure.
- Secure by Default: product should be secure right out-of-the box without having to change its default configuration.
- Secure in Deployment: product should be accompanied with the necessary administration guides and management tools to help the customers to use it securely.
- Communications: there should be effective and efficient communication channels between the development organization and the customers so that any problems found in the field can be addressed promptly.

The SDL is largely based on the proven security practices and principles used in IIS 6.0 but it has also introduced the following key improvements since then (Lipner and Howard, 2005):

Education

Instead of educating people at the last minute during the security push, now all the personnel involved in software development are required to take annual security training, whose contents are frequently updated by SWI team with the latest research results and findings in the field.

Organization

The SWI organization has now expanded into a multi-functional organization, which includes SWI security buddies (acts as a security advisor to the development team, has veto power for product shipment), attack and penetration test team (does whitebox testing at product level, conducts code review in critical areas of code), security tools development group (develops and maintain security related tools such as PREfix and PREfast), strategy group (updates SDL every 6 months, provides security training), Windows Vista penetration test team (concentrate security testing and code auditing on Windows Vista).

For teams of large products or products facing significant security threats, they now have their own dedicated security PM and penetration testers.

Development Process

Product planning documents and functional specifications are required to contain security considerations.

Secure design and testing guidelines published by SWI are being used across product teams. Threat modeling has become the indispensable tool for product design as well as product testing. The test team uses the threat model developed by the design to identify key data entry points and high risk areas, and then performs targeted whitebox and blackbox testing as opposed to the previous ad hoc testing.

The steps of security push have been integrated into the product development process. In the meantime Final Security Review (FSR), which occurs two to six months before the product is released, has been introduced to assess the overall security of the product. FSR is performed by the SWI security buddy with assistance from the SWI attack and penetration team as well as the product team. The assigned SWI security buddy conducts his independent review of the product and gives his security assessment to the product team and the top management. His main activities consist of interviewing the development team to evaluate their process compliance to SDL, examining the security bugs that are deemed false alarm by the product team, and reevaluating the product design in light of the latest security trend in the field. If it is a major software release, the security buddy will ask SWI penetration team to perform additional penetration testing. At the security buddy's discretion, he may also introduce external security consultants to carry out independent security reviews.

Security Response Process

MSRC has evolved its Software Security Incident Response Process (SSIRP) over the years. Having learnt the importance of getting all the parties into one location from the 2003 Slammer Worm incident, SSIRP has been improved so that once a major virus outbreaks or "zero day" disclosure of significant vulnerability is observed in the field, all the internal security experts in the affected areas are paged to convene on a phone bridge. If the problem is confirmed by this group, two teams will be formed to pursue the resolution. The emergency engineering team is composed of security experts and representatives from the product team with their main focus on finding the technical solution. The other communication team is responsible for passing along the engineering team's efforts to the outside stakeholders such as the press, customers and government agencies. In the meantime, senior management are kept in the loop and informed of the progress (Charles et al, 2006).

3.4 Observations

Microsoft's SDL is a lite implementation of the SDLC framework that we reviewed in chapter 2. Its successes have not only validated our theory that true software security can only be achieved via incorporating it into SDLC, but also demonstrated how to effectively introduce such changes into the development organization:

Although Microsoft has incorporated many key recommendations from the pundits into each stage of its development process, it heeds not to make the process overbearing. Unlike many other corporations where new processes are decreed from the top management and pushed down the throat of the engineering teams, Microsoft didn't introduce its SDL until the completion of corporate wide security awareness training and the successes of a series of pilot programs. After all, a process can dictate "who" does "what" at "when" and "where" (i.e. the kind of activity that someone has to do at certain stage and time) but can not control the "how" (i.e. the quality of how people perform the prescribed activities). If people are not convinced and motivated by the "why" (i.e. the rationale behind the process), the process will just create piles of meaningless paper trail, reduce employee productivities, and produces discontented employees while marginally achieving the intended result. Microsoft seems to be well aware of such phenomenon and have taken a series of incremental steps to introduce the new process to its development organizations.

The origin of the SDL can be traced back to the Windows 2000 era, where only a dozen or so SWI experts and penetration testers were conducting security testing and auditing of the Windows code base. Their activities were behind the scene and didn't interfere with the product team's development (unless security bugs were found). As the company began to realize the magnitude of the security issues it had at hand, it started to assign security experts to the most security critical product teams to advise them on security issues. As such approach turned out to insufficient, they rolled out massive security training with strong commitment from the top management including personal email message from Bill Gates. They not only well elaborated the rationales behind their security initiatives but also succeeded in conveying this business need into an engineering challenge that the developers can resonate and desire. For example, Howard and LeBlanc (2002) wrote in their book that the developers were so excited about security that one development lead in the IIS 6.0 team even offered a plaster mold of his pinky finger to anyone discovering a buffer overflow defect in his code – the core web server engine. And many did respond. Even with such well received responses from the engineering, the corporation didn't start to make dramatic process change. Instead, it selected key security design principles such as threat modeling and code review, and tested their effectiveness within several Windows product groups. It was only after these measures had proven themselves through the reduction of vulnerability reports in the field that the SDL was formalized and mandated. The SDL is also designed to be as least burdensome to the engineering as possible. Developers were equipped with automated security tools (e.g. PREfast) to ease their work. Xxx All of these built up the momentum and paved the way for the successful introduction of the SDL.

When being asked about which security practice in the SDL was the biggest bang for the buck, the SDL team pointed to threat modeling (Lipner and Howard, 2005). Although threat modeling is crucial in the sense that it serves as the foundation for product design, implementation and testing, I think the performance of the SWI organization is equally important for the success of SDL. Not only the product teams rely on its security training and advice to conduct effective threat modeling, but the corporation also depends on it to conduct independent reviews and provide product security assessment. Furthermore it has the responsibility to monitor the security landscape in the field, feed the latest research results and industry practices back into the organization, and update the SDL accordingly. Given the complexity of software security and its potential conflicts with other product requirements, having a dedicated organization whose main purpose is to improve the security across all product lines is not only necessary for providing the aforementioned functions but also pivotal for the necessary check and balance (i.e. ensure security has been properly handled on both the product level and the firm level).

In essence, threat modeling together with the competent SWI organization has led to the success of SDL hence the significant security improvement of the products.

3.5 Recommendations

- Actively pursue security input from the users and educate them about security. Many teams in Microsoft are development driven, which by itself is not necessarily bad. However, security is so sensitive to the environment that a small group of smart engineers alone can not think out all the possible attack scenarios and we can not afford the trial and error method used for functional features. While I am doubtful of the effectiveness of the requirement engineering approach proposed in chapter 2, we at least can do the following:

    - Make threat models available to lead customers, security product vendors and researchers. Encourage their inputs esp. on the design assumptions and risk assessments;

    - Provide well documented APIs and "user innovation toolkit" so that the customers can tailor the product to their own security needs (Franke and von Hippel, 2002).

- Educate the developers about the weakness of threat modeling to prevent complacency. While threat modeling is a powerful tool for the developers to understand the security implications of the system, it is not panacea. Such a chain-of-event model can not address the following failures (Leveson, 1995):

- System failure resulting from dysfunctional interactions between the modules at run-time;
- Human errors. For instance the operator could be confused by the security design or the security related operational procedure and mistakenly configured the system wide open.
- Adaptation. Software as well as its environment evolves over the time. Therefore threat model may become obsolete unless being actively maintained;
- Unknown attacks. Attack tree is good at modeling known attack methods/paths but not the ones unknown at the time;
- Common cause failures. This usually occurs when people don't validate their assumptions or validate them enough. For example a designer may mistakenly think object A and B will not be breached at the same time hence construct them in adjacent layers in order to defend the system in depth. However, both A and B depends on another class C. Hence breach in C will cause both objects to fail and so it is with the entire system. The problem is especially hard to discover when A, B and C are all complex components/products belonged to different product teams.

• Develop better threat modeling tools to:

    - Link components' threat models together to produce the system level threat model. Currently threat modeling is done at the individual component level. For instance, IIS is just one component of Windows Server Operating System. Although individual component's design documents including threat model are all stored in the central spec depot where people working on interdependent components have the access, such manual coordination is inefficient and error prone esp. as the software inevitably evolves over the time. In addition, just as local optimization often does not result in global optimization, methods to secure individual components may not be the best practices for the entire system. A system level threat model can help developers to identify these problems early on;

    - Inspect the security assumptions and dependencies across the different components to ensure they are sound and consistent. And when a new attack pattern emerges in the field or some design assumptions change or a new security vulnerability is found, the tool can automatically go through the corresponding threat models and then pinpoint the affect components;

- Develop a corporate wide knowledge system to share common threats and the corresponding mitigation methods. Product teams can readily reuse them without reinventing the wheel.

- Emphasize usable security during product design. Research has shown most of the security incidents were caused by human error. Usable security is not just about GUI. It is also about the correct design of control and data flow. They must be considered together during the design. Any security feature requires user interaction should provide a GUI prototype and have it validated with some targeted customers. Although Microsoft teaches secure user interface design as part of SDL, many of its more development driven product teams unfortunately do not seem to realize its importance.

4. Case Study: Apache HTTP Web Server

4.1 Brief history

Apache HTTP server was originally developed by Rob McCool at the National Center for Supercomputing Applications (NCSA) in University of Illinois at Urbana-Champaign, and known as NCSA HTTPD. Soon after Rob left NCSA for Netscape in mid-1994, a group of volunteers, who called themselves the "Apache Group", teamed up to pick up his work and started to support and maintain the sever code. In less a year their new version overtook the now defunct NCSA HTTPD and became the leading web server on the Internet. The server has since evolved from a UNIX only application to a cross-platform product. According to the latest report by Netcraft Web Server Survey, Apache Web Server has remained as the most popular and dominant web server on the Net since mid-1996 (figure 13).

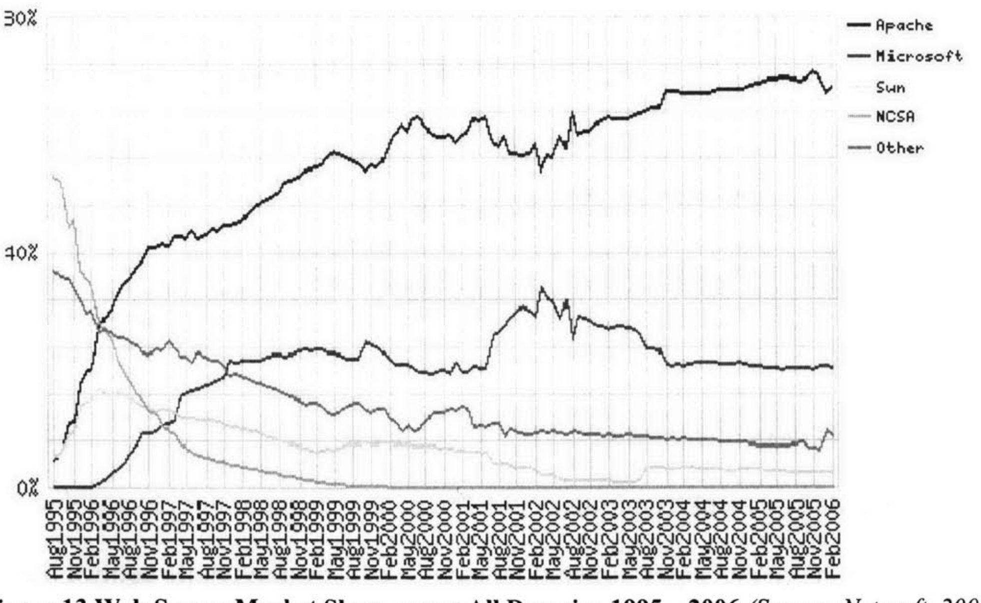


Figure 13 Web Server Market Share across All Domains 1995 – 2006 *(Source: Netcraft, 2006)*

As the Apache server project got bigger and new web related sister projects were created, the Apache Group created the non-profit organization Apache Software Foundation (ASF) in 1999. The purpose was to facilitate and coordinate these different programs as well as protect individual contributors from legal claims directed at the projects. However, each project retains the authority in its product development and has great degree of freedom in defining its own governing rules.

4.2 Organizational structure[12]

With a collaborative, consensus-based culture, the foundation does not have a hierarchical organizational structure but the following different roles and responsibilities:

- User: anyone that uses the product. Their active participations to the project range from filing online bug reports or feature enhancement requests to answering other users' questions in the support forum or mailing lists.
- Developer: a user that develops code or documentation for the product. Their contributions (usually in way of patches) must be reviewed and approved by the committers.
- Committer: a developer that has the privilege to make changes to the code repository and has signed the Contributor License Agreement. They are the main work force developing the product.
- Project Management Committee (PMC) member: a committer elected due to his technical merit and commitment to the project. They as a whole control the destiny of the project including feature sets, release schedule etc. Each PMC has

[12] This section is mainly based on information from http://httpd.apache.org/dev/guidelines.html

at least one ASF officer as the chairperson who oversees the day to day operations of the project and report to the ASF board.

- ASF member: a person elected by current ASF member for his contribution and commitment to the corporation. They legally represent and care for the foundation, have the right to elect the board, and propose new projects. They may also serve in the individual PMCs.

The Apache Web Server project is a meritocracy - one advances his role through his technical contributions to the project. Everyone has the read access to the code and can propose changes. However, only committers have the write access to the repositories.

A developer can become a committer if he has

- Provided consistent and technically sound contributions to the project for over six months, and
- Nominated by an active PMC member, and
- Received unanimous approval from all the active Apache PMC members.

The committers take care of the technical details of the project by casting their binding votes on any technical issues. No changes can be made to the code or document repositories without their approvals.

Selected committer will be invited to join the Apache PMC upon unanimous approval from all the active Apache PMC members. The committee as a whole manages the operational aspects of the project such as resolving licensing issues related to the project, nominating new committers or PMC members etc.

Due to its geographically dispersed user and developer communities, the project is managed and communicated mainly through online mailing lists (public and private).

Most of the project coordination is done via numerical voting and consensus gathering. Everyone votes in one of the following flavors:

- +1, a yes vote.
- 0, abstain, or would like to pass this onto the others to decide.
- -1, a no vote. On issues where consensus is required, this vote is a veto. Veto can not be overridden although the voter can rescind it should he changes his mind. Veto must be accompanied by a detailed explanation.

Although every developers are welcomed to vote and voice their opinions on any issues pertaining to the project, only the votes from the committers count (i.e. binding vote).

Motions such as proposing a new feature or making major changes to the code require *consensus approval* (i.e. at least 3 binding +1 votes and no veto). Action such as making a public release requires *majority approval* (i.e. at least 3 binding +1 votes and more +1 votes than -1 votes). All other proposals such as release plan, showstopper[13] designation,

[13] Critical bug fix or new feature that has to make into the next public release.

product changes are considered to have *lazy approval* until someone votes -1. Then the group will have either a consensus or a majority vote to resolve the issue.

4.3 Development Process[14]

Like many other open source products, Apache does not have a formalized development process. Instead, each developer iterates through the following series of actions:

- Search the Apache Bug Database (i.e. Bugzilla) and identify an unassigned problem that is interesting to him. The problem could be either a bug or a feature request. It could be filed by other users or the developer himself;
- Develop and test the solution on his own machine;
- Edit the original bug report, specify "PatchAvailable" in the Keywords field, and then attach the patch to it;
- Send an email to the developer's mailing list with subject line prefixed with "[PATCH <PR-number>]" to solicit code review. This step is optional if the changes are not significant and the developer is also a committer;
- Commit the code and the corresponding documentation changes to the repository. If the developer is not a committer, the changes need to be approved and then submitted by one of the committers.

[14] This section is mainly based on information from http://httpd.apache.org/dev/guidelines.html and http://httpd.apache.org/dev/release.html.

Note: *Consensus approval* is required even for a committer if the changes are significant or doubtful, or modify system level behaviors such as the size of the program, semantics of published APIs etc.

Any committers can volunteer to be the Release Manager (RM) and make a public release. He is free to decide the timing as well as the content of the release. Although it is recommended that the RM inform the developer community ahead of time about his release plan, it is not required. Pending release has no effect on the current development work although people may voluntarily refrain from making last minute changes. The PM may perform sanity tests on the candidate release. Executing httpd-test suite and having the candidate running on Apache.org for 48-72 hours are highly recommended but the PM can choose whatever tests he deems appropriate. Once the PM is confident in the code, he can release it with the alpha designation. Then it is up to the entire Apache community to test and determine the fate of the release. If the release proves to perform the basic functionalities and has received majority approval, it will advance to beta designation. Later when the community is confident about its quality and gives majority approval for its consumption by the general public, it receives the General Availability (GA) designation and is released to the public. Not all the releases will make to GA, though.

4.4 How is software security being handled?

"Security as a mandatory feature" is one of the main principles of the ASF and shared by all its projects. However, there does not appear to be any guidelines, processes, or tools in place to help the developers consider the security aspects of their design. The project rather trusts the developers use their own methodologies and tools to come up with secure code, the peers do thorough security review, the user community test and promptly report vulnerabilities, and the developer community quickly response and mend the security holes.

To expedite the resolutions of security defects and prevent malicious users from exploiting them, ASF asks users to report all security problems related to its products to the Apache Security Committee (ASC) before publicly disclosing them. The ASC is composed of six senior members[15] whose knowledge span across many Apache products. The team has a well defined process which is depicted below (Shapira, 2006):

Security incident is first reported to the ASC via an email to security@apache.org, which is monitored by all the committee members. After validating the issue, ASC forwards the relevant information to the corresponding product team for further verification. Once the product team confirms the problem, the appropriate developer will be contacted to work on the solution. In the meantime ASC will inform external security tracking parties such as CERT and CVE of the vulnerability. After the fix is created and verified, the product team issues a new release if the vulnerability's rating[16] is critical or important. Otherwise a patch will be announced and made available for the public to download. The original reporter and the ASC are kept in the loop during the entire problem resolution process. All their communications, however, are kept in a private mailing list.

[15] See their list at http://people.apache.org/~jim/committers.html. They are the committers whose project columns contain "security".
[16] ASF defines its vulnerability ratings at http://httpd.apache.org/security/impact_levels.html.

4.5 Analysis

Based on the previous observations, one might be enticed to conclude the Apache Web Server project takes the traditional "penetrate-and-patch" approach to tackle software security. However, if we look closer at the different development activities, we will get a quite different story.

Let's start off at the specification stage. Software is developed to solve a particular problem. One main issue for large software project is the development organization's lack of knowledge in the problem domain (Curtis et al, 1988). To mitigate that obstacle, traditional software engineering doctrine suggests the involvement of all stakeholders including the lead users in the specification phase and companies create product marketing and management teams to better understand customer needs and translate them into the engineering specifications understood by the design teams. However, because the customers often either do not understand their own needs, or fail to articulate them clearly to their vendors, or evolve their needs over the time, the best requirements we can get are usually incomplete and often are moving targets. To make the matter worse, as the requirements being translated and traversing through the organization, the information is not only delayed but often lost or distorted (figure 14). Due to those limitations, many Open Source Software (OSS) advocates argue that despite the lack of formal specification, OSS developers have better understandings about customer needs than the commercial software vendors because they themselves are the users. While the validity of such generalization is up for debate, researchers do observe the fact that people who make the most contributions in the Apache projects are the lead users and they are highly skilled (Franke, 2002). Furthermore due to the fact that the Apache Web Server community mainly consists of webmasters, system administrators, and service providers, who have vested interests in the security of the server, there are usually quality discussions about the security aspects of new features. Adding up together, these practices seem to have more or less compensated the lack of clearly analyzed and defined security specifications.

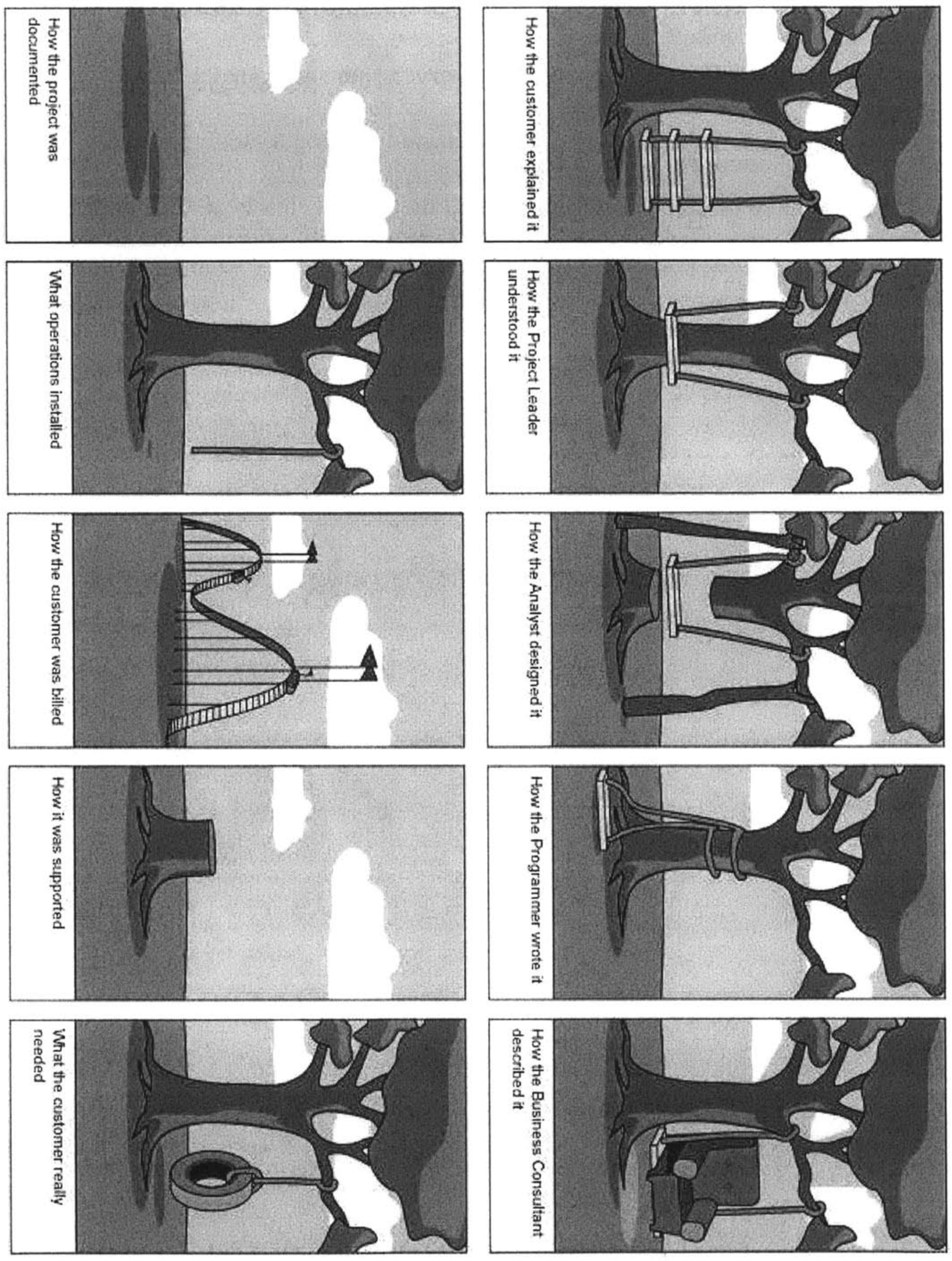

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Figure 14 Software Project Management Cartoon (*Source:* *http://www.jacobsen.no/anders/blog/archives/2004/05/17/the_tire_swing_cartoon.html*)

Speaking of design, the Apache Web Server has a modular architecture from the very beginning (Feller and Fitzgerald, 2002). It keeps the basic functionality that all the web sites need in the core module but leave the bells and whistles to other separate components (Mockus et al, 2000). It is also designed to run on non-privilege account. These two designs forms the secure foundation for the Apache Web Server and stand in sharp contrast to IIS (prior to the latest version 6.0) which had been hit hard by a slew of exploits due to its monolithic architecture and dependency on the system account. However, such foresights do not appear to be the fruits of careful security considerations but of good engineering principles, vision in product extensibility, and the need for parallel development. As a matter of fact, the project leaves the design and implementation work (including the security aspects) to the individuals and relies on their professional expertise as well as the peer review process to ensure product quality. This seems to be the application of the famous motto of "given enough eyeballs, all bugs are shallow" (Raymond, 1999). Many pundits consider such claim as a myth in that having source code available do not mean people will automatically examine the code and more important if there are properly trained people looking at the code from the perspectives that you want (Viega, 2004). However, a careful examination of the Apache Web Server project reveals it is precisely this principle leading to the high security of the server:

- There are plenty of external eyeballs looking at the code from security point of view. Shapira (2006) found that external users report more than 70% of the security bugs. And security analyst firms, companies providing web hosting services, and individual security analysts and hackers are the main reporting sources.

- These eyeballs are well versed in security. The security analyst firms are the most valuable inspectors because they have the most in depth knowledge about software security. Many of them use Apache Web Server as the reference model for their security test tools, therefore they routinely inspect new sever releases and test them for conformance and new vulnerabilities (Shapira, 2006). As for the 2nd type of the inspectors, the web portal providers, they have the inherent incentive to audit the code and run tests to ensure the security of the server, although they are not as dedicated and knowledgeable as the security firms. The 3rd category of the inspectors is the security minded individual. They often do the code auditing and testing for personal reasons such as glory and fame, or providing advertisement for their personal security consulting services. They have good skills and often provide fresh security insights to the product team.

- The internal developers also take code review seriously. Mockus et al (2002) found most of the core developers review all the changes. And comments made from the wider communities are often also accepted into the changes (Mockus et al, 2002).

- The vast majority of the features are developed by the most skilled developers. Mockus et al (2000) observed the fact that the top 15 developers contributed to most of the new functionalities while the community at large is more active in providing bug fixes. This ensures the overall quality (including security) and stability of the product.

- Most security bugs are fixed by most senior members in the product team and reviewed by the whole developer community (Shapira, 2006). This helps to reduce the probability of security fix introducing regression errors and improve community learning.

Although system level testing is delegated to the entire community, Mockus et al (2000) discovered that the defect density before system test is much lower than that of commercial software. They attributed it to either the high skills of the developers, or the effectiveness of the code review, or the diverse background of the developers (Mockus et al, 2002). The dependency of community testing is particular interesting under the security context. As we have discussed in the literature review section, security is an emergent system property. This means many difficult security defects are very subtle and often arise from dysfunctional interactions that occur only under certain conditions. Such problems can not be identified from design analysis or code review but through thorough testing. However, a company can only afford to test a set of common customer scenarios and hence let bugs manifested under other conditions slip through. By having such a heterogeneous and highly capable community acting as its testers, the project dramatically extends the test coverage of its code hence increases its chance to uncover and fix those peculiar vulnerabilities early on and improve the security of the product.

In terms of maintenance, Mockus et al (2002) observed Apache Web Server project has faster problem response time than that of commercial software vendors. Patch becomes publicly available almost as soon as it is being created. This stands in sharp contrast to the common approach used by commercial vendors - the security fixes are usually bundled into new releases, which are made available only at a predetermined time. Since one can never build 100% secure software, turnaround time is critical to prevent the vulnerability from being exploited. This "release often, release early" approach builds up trust in the user community and reduces the overall security risk of using the product.

From organizational point of view, the Apache security group and its product team have much more diverse background than those of the traditional software companies. For example, the security team consists of members coming from secure web site hosting company, Linux provider, cryptography research lab, and secure data center provider etc (Shapira, 2006). The variety of their backgrounds ensures different security perspectives being considered during product development and cultivates healthy knowledge sharing and synthesizing within the community. Furthermore, their individual interests are well aligned with the security goal of the project. To many of them, security is not a nuisance that one has to deal with but a technical challenge that they love to solve.

4.6 Recommendations

The magic ingredient of the security achievement of Apache Web Server is its people: they are not only highly motivated and skilled in producing secure code, but also have diverse background with different security viewpoints. The community has also attracted enough of them to make its peer review process effective, which extends through all stages of the product development lifecycle – feature proposal, design and implementation, testing, releasing, and maintenance. Unlike in the traditional software companies, where only a limited group of people participate at each stage and the decisions are often made in haste, everyone can voice their opinions at any time in Apache and many have been well received by the developers (after all there are no scheduling or cost constraints for OSS projects). This is particular important for security – software security is a difficult subject that the industry as a whole has just started to pay attention to and in desperate need of new ideas. Many active participants in Apache Web Server project are the lead users whose security needs are far in advance of the regular users. Their opinions and proposals hence may become the innovations that eventually spill over to the general users (Franke and von Hippel, 2002). The peer review process presents a unique nurturing platform for such security innovations to be heard and applied.

However, there are still weaknesses in the project's development process that could really use some improvement. Below are a few recommendations.

- Perform detailed security analysis during system design. Comparing with the earlier versions of IIS, Apache's security advantage really arises from its original architecture – modular design, separation of privileges and open architecture, all of which are sound software engineering techniques which also happen to be good security practices. In other words, the security success of the project is by and large the dividends from their good general software engineering skills. While those are suitable for fending off old type of amateur attackers, they may not continue to be sufficient to protect against increasingly sophisticated cyber criminals. Security holes such as the recent cross-site scripting vulnerability CVE-2005-3352 is almost impossible to be spotted during code walkthrough and the modular architecture doesn't mitigate the issue either since the problem is in one of the base modules (i.e. loaded by default). Furthermore, individual developers may have a good understanding of the security risk that he is facing, but they may confuse the tree for the forest. As the result, a solution appropriate for one occasion may well open an attack avenue under another context. Therefore the community needs to adopt some types of security analysis such as attack patterns, threat modeling etc to carefully examine existing architecture as well as new feature proposals to understand their holistic security implications and design accordingly.

- Provide tools to detect implementation vulnerabilities. Although code reviews in the Apache community are very effective in that very few the all-too-common ancestor-of-all-security-holes buffer overflow bugs have been found in its code, there are occasional overlooks such as the most recent NULL pointer dereference flaw CVE-2005-3357. Unlike functional bugs, many implementation vulnerabilities are often very hard to discover by human. But once they become well understood, often tools can be made to automatically detect them. Therefore maintaining and mandating a set of baseline code analysis tools to be executed prior to any code submission will really help to improve the security of the product. This is especially important since the hackers may well be using them to scan the code for exploits!

- Coordinate security testing. The project relies on the community at large to provide testing coverage for the product. Although it has its own regression test (httpd-test project) and stress test (flood project) suites, they are neither mandated nor security focused. The project has no security testing guidelines and the volunteers testers are free to test the product with whatever methods they like. Although this promotes community building as new comers often email the tester distribution list for guidance as well as the diversification of testing, it also introduces certain degree of inefficiency since there is no coordination of testing. The inefficiency does not come from the fact people overlapping their testing areas, which from security point of view is a good thing since it provides the complete coverage from all possible angles. But rather because people don't know what areas each others are testing, they may end up with over testing the common areas but under testing or missing out the less common ones. This is not a big issue for functional testing since those under tested are usually features that are less used anyway. However, such mentality is harmful for security since the security of a system is determined by the security of the weakest link. As long as there is a hole in the code, regardless of its usage pattern, it will be exploited and the entire system being compromised. Therefore the community should set up a centralized database or forum (e.g. wiki page) that shows the tests being accomplished alongside with its test methods and code coverage, so that volunteers can make informed testing decisions and the project management can get a better sense of the overall testing quality of the product.

- Document system level designs. This may not be an issue for regular feature development. After all, most of the features are developed by about a dozen highly skilled core developers (Mockus et al, 2000). They know the code inside out and don't need the documents. For the rest of the community who contribute the patches, since the project is governed by meritocracy, they are expected to prove their way up by reading the code. Therefore by not having design documents, the project sets up a bar to discourage the less proficient or committed people from joining the project and hence maintains the high skill level of the community. However, this practice is counterproductive for security. As we have discussed in the previous literature review section, security bugs are often subtle and manifest themselves only under certain rare conditions. Therefore the original authors of the feature are probably not the right one in detecting the security issues and static code review won't help much in this regard either. But without the proper design documentations describing the architectures and design rationales behind the features, it is very difficult for the wide community to help them out. After all, the most effective testing method to uncover hidden security holes is targeted penetration testing, where the testers use the design of the feature as the guide to identify the theoretical weak points, and then launch attacks against them. Without adequate design documentation, average testers just can not use this method. And we end up with a small pool of testers hence a product without enough test coverage.

5. Conclusion

Having surveyed the academic researches and recommendations in chapter 2, its de facto implementation at Microsoft in chapter 3, and the different approach taken by the Apache HTTP project in chapter 4, let's revisit the following key questions that we have raised in chapter 1:

- What should the software development organizations do in order to produce secure software?
- Since it is unlikely one development method will fit all projects, what are the pros and cons of them and their application domains?

For the first question, the answer should be quite evident by now – we can only achieve true security in software if we integrate security into the entire software development life cycle.

The solution to the second question, however, is not so straightforward. The framework we reviewed in chapter 2 recommends an all-encompassing secure software development process. However, our case studies have demonstrated two dramatically different approaches while both have achieved very good results. As a matter of fact, in both cases we see the emphasis on the beginning phases of the development process, namely the requirement gathering and high level design as opposed to the latter ones such as implementation and testing. In addition, the complexity of the development process is decreasing from the suggested framework to Microsoft's SDL to Apache. These thus seem to have confirmed our hypothesis that although security should be "designed in", the actual development approaches can vary according to the business context.

Although there is no single silver bullet to solve the software security problem, based on the insights from the previous chapters we believe software vendors should take the following actions as *minimum* requirements to improve the security of their products at the same time pick and choose the other methods and practices introduced in earlier chapters as the supplements.

- Understand your business and its security implications.

  Security should be considered within the business context (Anderson, 2001). No one can make a piece of software absolutely secure without making it absolutely useless at the same time. And there are many situations that we don't need very secure software anyway. It is all about calculated risks and benefits. Businesses need to examine the market requirements, the competition landscape, government regulations, and public perceptions in term of security for its products and make a conscious decision to determine whether and how much security they need. Coming from the engineering background, I would love to make secure products at all costs. But at the end of the day, if what you produce doesn't make economic sense, all these engineering efforts will go down the drain.

- Educate and prepare your teams.

If you have decided you need security in your products, you will probably have to make changes in your engineering processes and very often the business ones as well. Such changes will inevitably move people out of their routines and comfort zones. And “human nature resists that which is not understood” (Eckes, 2003). Management must involve and educate its teams using the languages and goals that they can understand and relate to. For business people, for example you may want to show them the economic and cash flow analysis in support of your security initiatives. For the technical-minded, maybe give them a live demonstration on how the hackers can break into their products and try presenting security to them as an engineering challenge that geeks would love to solve.

Once you have mobilized the mass, the next thing you need to do is providing security training to the people involved with product development. You don’t have to retrain every PM, developer, and tester, although this will be highly desirable. You can concentrate the training on the key players such as the lead PMs, chief architects, and the lead testers. And customize the education towards the different roles and skill levels of the audience. You can then delegate the training of the rest to these personnel. If you can not afford outside consultants to provide the coaching, buy some security books for your people. Software security requires a different skill and mindsets that many people don’t have. They must be trained properly.

On the organizational front, you should consider to have a senior engineer (preferably a small team of PMs, developers and testers who are dedicated to product security) as the in-house security expert who is dedicated to monitor the security landscape, act as the security resource to the rest of the organization, review and coordinate the security efforts of the organization. Experiences from Microsoft have shown their SWI organization is pivotal in the overall security improvements to the company's products. And we agree with their approach in this regard.

All in all, engage and get the buy in from your people and guide them to make the aforementioned changes. Do not decree from the top without significant agreement from foot soldiers. After all it is them who eventually make the product, processes are just means to an end. Without their cooperation, you will just create another layer of bureaucracy.

- Get your security requirements right.

If you are an OSS vendor with a large user and developer community, congratulations since you will usually have a more intimate knowledge of the problem space than the commercial vendors. Regardless of the vendor types though, you should engage your customers to understand their security needs. It is likely that they may not give you specific instructions on their security requirements since they themselves may not know them well either. But even if there are seemingly good security requirements, you still want to have an in-house security expert or lead developer to go to the customer sites, observe how their business operate to validate the claims. After all, security is highly sensitive to the environment which usually is difficult to explain in words. You should also derive security requirements from the related government regulations and the competing products, as well as the inputs from your in-house security expert about the latest security trend in the field.

- Design secure and resilient architecture.

In addition to apply the security design principles in chapter 2, you should do a high level security analysis of the product architecture. Methods such as thread modeling, attack patterns are very helpful for this purpose. However, since these techniques are all passive in the sense that they only take care of existing attack models, you need to design mechanisms to aid speedy discovery and remedy of new vulnerabilities after the product is released. Features such as the following should be considered in the design:

  - Monitoring, reporting and logging. Not only we need these capabilities for speedy problem detection and troubleshooting but the law enforcements also depend on them to effectively investigate the crime and prosecute the perpetrator (Kshetri, 2006).
  - Maintainable architecture. Product should automatically notify its users of new security patch and offer the choice of installing it (preferable hitless);
  - Since the users will likely to have different level of security needs depending on the context, you should consider providing them with customizable

security toolkits that they can add on top of the product. In other words, make your users be your security developers;

  - Apply user-centered design principles (Sasse and Flechais, 2005). UI and operational procedures for the security features should be prototyped and confirmed by the lead user groups.

The security considerations should be clearly recorded in a separate chapter of the high level design document including the rationales, assumptions you have made for the current design as well as the other alternative solutions that have not been taken (Whittaker and Atkin, 2002). And this chapter should serve as the guide for the following implementation and testing efforts.

- Assure the code is properly implemented by having developers review each other's code. Special care should be taken in implementing and reviewing the security critical section of the code.

- Test the product according to the security analysis chapter in the high level design spec. You should have some dedicated security testers who are creative and can think the way that hackers do. They should use the security analysis as the guide to conduct fuzz testing. It is best to have a diverse tester group. However, since it will likely be very expensive, you may be able to offload some of the work to your lead customers by offering them the product earlier for their lab testing. You should provide them with your high level security analysis and your system test plan so that they can review and better target their tests. The detection for past security vulnerabilities should be automated and included in the regression test suite.

- Include your security considerations as distinctive sections in the user documents. Security procedures are best to present in the form of context sensitive run-time help and should show the user the entire operational procedure.

- Design and put in place a security incident response process. It should be designed as light weighted as possible to facilitate speedy response to emerging threats in the field. Have a dedicated staff serve as a coordinator between the customers and the development organization to ensure the smooth handling of the security issues, drive the root cause analysis and ensure the organizational learning (Argyris, 1976) from the incidents.

- Have some dedicated staff to actively monitor security landscape and feed the learning back to the organization.

The above framework is heavily tilted towards the early stages of the development cycle, i.e. the requirement gathering and design stages, mainly because of our conviction that an insecurely specified or designed system can never be appropriately reversed or remedied at implementation or testing stages. On the contrary, we believe those flaws will only be magnified and worsen in later phrases. Furthermore our two case studies show they provide the most return and their success are the foundations for the rest phases.

We have not put much emphasis on the tools in our recommendations because they are not only still in its infancy with either high noise-to-signal ratio (i.e. high false alarms) or steep learning curves (i.e. need to use the tool's language to instrument the application's code) but effective to merely detect known and relatively simple vulnerabilities. We feel appropriate training for the developers and testers can bring more security to the product than them for now. The exception is the automated security regression tools. Regression is particularly important for security since security issues are often so subtle that it is very likely one small change in one place can reintroduce vulnerability in another remote location without the developer's knowledge. In addition, since attackers are usually armed with ready tools to exploit past defects and routinely execute them on their targets, the consequence of reopening up an already well-known security hole is much graver than that of a new vulnerability.

The bottom line: motivate and educate your people on security, then guide them to incorporate it into their development process with some help of security tools.

6. Future work

The task force report from the conference of the National Cyber Security Summit in 2003 states the following (NCSP, 2004):

> "No simple silver bullets will solve the software security problem. As a long-term multifaceted problem, it requires multiple solutions and the application of resources throughout the lifecycle. Improving software security and safeguarding the IT infrastructure is a research and education issue for universities; a skill, process, and incentives issue for producers; a requirements issue for customers; a quality and testing issue for providers; a maintenance and patching issue for IT administrators; an ease-of use issue for users; a configuration issue for installers; and an enforcement issue for governments."

In this thesis we focus mainly at the issue from the software producer's side. However, just as the report has indicated, software security is really a system problem that requires the coordinated efforts from all the stakeholders, not only the software vendors.

One interesting observation from the IIS case study is that most of the design philosophies, which have been attributed to its great security improvement: separation of privileges, disabling of the unused functions, modularity design (being used in the upcoming new release IIS 7.0), are hardly new ideas. They have been taught in the universities and advocated by the academia since the 1970s. Engineers in Microsoft are renowned for their tech savvy hence it is hard to believe they don't know about them already. Then why had they kept building monolithic and gigantic products until recently?

In contrast, the same design philosophies have been long adopted by OSS. The diverse needs of their users have led to the more extensible hence modular design of OSS products (Fielding and Kaiser, 1997; Narduzzo and Rossi, 2004). This has then in turn given the users the ability to tailor their product to their own security needs whereas for commercial ones they have to accept whatever the vendors have given them (Cowan, 2003).

Many pundits have therefore claimed it is not that we do not have the technologies to tackle the security problem, but the lack of will from or incentives for the vendors. They argue that due to the high network externality and its lock-in effects in the software business, getting the product out of the door with the most feature sets has trumped the quality concerns. There have been computer simulations that show market leaders can maintain their lead even if their products are inferior to their competitors' (Zipkin, 2005). Real life examples such as IE vs. Firefox, where the latter has failed to make a dent on the former market share despite its feature and security advantages[17], seem to have backed up this theory. So is this a market failure? Should security be tackled by the government through polices and legislations? Or should we let the market take its course?

Regardless of the approaches you take (i.e. government intervention or the invisible hand), in order for any of them to succeed they all need one thing in common. That is the

[17] Details at http://techrepublic.com.com/5208-9592-0.html?forumID=88&threadID=174011&start=0

customers should be able to evaluate the security of the alternative products and then make informed decision. However, security assessment for software is particularly difficult. Unlike functional requirements that what we don't know won't hurt us, it is precisely the vulnerability that we haven't thought about during the design will bite us. How do you look for something you don't know even if they exist?

Furthermore, software products are becoming inevitably complex and dependent upon each other. It is no loner sufficient to just have a secure operating system. The entire end to end solution needs to be secure. How to make the whole system secure while at the same time keep them useable and maintainable? How do you coordinate all these different vendors?

There are so many questions remaining unanswered. As long as we continue to push the industry to extend its technical envelop to achieve things that are inconceivable before (actually it is the beauty of this industry and the main reason why so many people love doing programming), and the market keeps demanding better, faster and cheaper products (not securer), I believe we are going to see more vulnerabilities, not less.

There is an old Chinese saying "the journey of a thousand miles begins with a single step". I think we have just started our first step into the journey for software security.

Bibliography

Argyris, C. (1976), *Increasing Leadership Effectiveness*, Wiley, New York, 1976.

Anderson, R. (2001), "Why Information Security is Hard - An Economic Perspective", in *Proceedings of the Seventeenth Computer Security Applications Conference,* IEEE Computer Society Press, pp. 358-365 [Online]
Available at: http://www.acsa-admin.org/2001/papers/110.pdf

Belani, R., Muckin, M. (2004), "IIS 6.0 Security", SecurityFocus.com [Online]
Available at: http://www.securityfocus.com/infocus/1765

Bishop, M. (1995), "UNIX Security: Threats and Solutions", invited talk given at the 1995 System Administration, Networking, and Security Conference, April 24-29, 1995.

Bishop, M. (2003), "What Is Computer Security?", *IEEE Security and Privacy*, 1(1), pp. 67-69.

Brooks, F. P. (1995), *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, Reading, MA.

Brown, M. (2003), "IIS vs. Apache, Looking Beyond the Rhetoric", ServerWatch.com [Online]
Available at: http://www.serverwatch.com/tutorials/article.php/3074841

Burke, B.E., Kolodgy, C.J., Christiansen, C.A., Hudson, S., Carey, A. (2005), "Worldwide IT Security Software, Hardware, and Services 2005-2009 Forecast: The Big Picture", International Data Corporation (IDC), Framingham, MA.

Charles, J. D., Kolodgy, J., Ryan, R. (2006), "Microsoft Security Response Center: The Taming the Chaos of Security Incidents with a Steady Process", International Data Corporation (IDC), Framingham, MA.

Cowan, C. (2003), "Software Security for Open-Source Systems", *IEEE Security & Privacy*, 1(1), pp. 38-45.

Curtis, B., Krasner, H., Iscoe, N. (1988), "A Field Study of the Software Design Process for Large Systems", *Communications of the ACM*, 31(11), pp. 1268-1287.

Cusumano, M. A., Yoffie, D. B. (1998), *Competing On Internet Time: Lessons from Netscape and Its Battle with Microsoft*, Free Press, New York, NY.

Eckes, G. (2003), "Make Six Sigma Last (And Work)", Ivey Business Journal [Online]
Available at: http://www.iveybusinessjournal.com/article.asp?intArticle_id=453

eMarketer (2004), "Scary Stats: Internet Attacks Worldwide", eMarketer, New York, NY.

eMarketer (2005), "The High Cost of Security", eMarketer, New York, NY.

Feller, J., Fitzgerald, B. (2002), *Understanding Open Source Software Development*, Addison-Wesley, Reading, MA.

Fielding, R.T., Kaiser, G. (1997), "The Apache HTTP Server Project", *IEEE Internet Computing*, 1(4), pp. 88-90.

Franke, N., von Hippel, E. (2002), "Satisfying Heterogeneous User Needs via Innovation Toolkits: The Case of Apache Security Software", MIT Sloan School of Management Working Paper # 4 [Online]
Available at: http://opensource.mit.edu/papers/rp-vonhippelfranke.pdf

Franke, N. (2002), "Characteristics of Innovating Users of Apache Open Source Software", University of Vienna Working Paper.

Frincke, D., Bishop, M. (2004), "Guarding the Castle Keep: Teaching with the Fortress Metaphor", *IEEE Security and Privacy*, 2(3), pp. 32–35.

Geer, D., et al (2003), "CyberInsecurity: The Cost of Monopoly", Computer Communications Industry Association, Washington, DC.

Howard, M., LeBlanc, D. C. (2002), *Writing Secure Code (2nd Edition),* Microsoft Press, Redmond, WA.

Howard, M. (2004), "Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users", *MSDN Magazine*, November 2004 [Online]
Available at: http://msdn.microsoft.com/msdnmag/issues/04/11/AttackSurface/

Jaquith, A. (2005), "Fear and Loathing in Las Vegas: The Hackers Turn Pro", Yankee Group, Boston, MA.

Kshetri, N. (2006), "The Simple Economics of Cybercrimes", *IEEE Security & Privacy*, 4(1), pp. 33-39.

Leveson, N. G. (1995), *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA.

Leveson, N. G. (2004), Class notes of "16.355 Software Engineering Concepts", Massachusetts Institute of Technology, Cambridge, MA.

Lipner, S., Howard, M. (2005), "The Trustworthy Computing Security Development Lifecycle", Microsoft Corporation [Online]

Available at: http://msdn.microsoft.com/security/default.aspx?pull=/library/en-us/dnsecure/html/sdl.asp

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

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

Mockus, A., Fielding, R. T., Herbsleb, J. D. (2000), "A Case Study of Open Source Software Development: The Apache Server", *22nd International Conference on Software Engineering (ICSE '00)*, pp. 263-272.

Mockus, A., Fielding, R. T., Herbsleb, J. D. (2002), "Two Case Studies of Open Source Software Development: Apache and Mozilla", *ACM Transactions on Software Engineering and Methodology*, 11(3), pp. 309-346.

Moore, A. P., Ellison, R. J., Linger, R. C. (2001), "Attack Modeling for Information Security and Survivability", Carnegie Mellon University [Online]
Available at: http://www.cert.org/archive/pdf/01tn001.pdf

Naraine, R. (2006), *Anti-virus Software: The Next Big Worm Target?*, eWeek.com [Online]
Available at: http://www.eweek.com/article2/0,1895,1913701,00.asp

Narduzzo, A., Rossi, A. (2004), 'The Role of Modularity in Free/Open Source Software Development' in *Free/Open Source Software Development*, ed. S. Koch, Idea Group Publishing, Hershey, PA.

[NCSP] National Cyber Security Partnership (2004), "Improving Security across the Software Development Life Cycle" [Online]
Available at: http://www.cyberpartnership.org/init-soft.html

O'Connor, T. (2006), "Cybercrime: The Internet As Crime Scene" [Online]
Available at: http://faculty.ncwc.edu/toconnor/315/315lect12.htm

Pescatore, J. (2006), "Augment Security Processes to Deal with the Changing Internet Threat", Gartner Inc., Stamford, CT.

Port80 (2005), "Port80 Surveys the Top 1000 Corporations' Web Servers", Port80 Software, San Diego, California [Online]
Available at: http://www.port80software.com/surveys/top1000webservers/

Raymond, E. S. (1999), "The Cathedral and the Bazaar" [Online]
Available at: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

Rosenberg, L., Hammer, T., Shaw, J. (1998), "Software Metrics and Reliability", 9th International Symposium on Software Reliability, November, 1998, Germany [Online]
Available at:
http://satc.gsfc.nasa.gov/support/ISSRE_NOV98/software_metrics_and_reliability.html

Saltzer, J., Schroeder, M. (1975), "The Protection of Information in Computer Systems", *Proceedings of the IEEE*, Vol. 63, No. 9, pp. 1278-1308 [Online]
Available at: http://cap-lore.com/CapTheory/ProtInf/

Sasse, M. A., Flechais, I. (2005), 'Usable Security' in *Security and Usability: Designing Secure Systems That People Can Use*, ed. L. F. Cranor and S. Garfinkel, O'Reilly, Sebastopol, CA.

Schneier, B. (2000), *Secrets and Lies: Digital Security in a Networked World*, John Wiley & Sons, New York, NY.

[SI] Security Innovation (2005), "Application Security by Design", Security Innovations Inc., Wilmington, MA [Online]
Available at:
http://www.securityinnovation.com/pdf/Application%20Security%20by%20Design.pdf

Shapira, Y. (2006), "A Threat-Rigidity Analysis of the Apache Software Foundation's Response to Reported Server Security Issues", S. M. Thesis, System Design and Management Program, Massachusetts Institute of Technology, Cambridge, MA.

Souza, R.D., Adachi, Y., Kavanagh, K.M., Parveen, K. (2005), "Forecast: IT Security Services, Worldwide, 2003-2009", Gartner Inc., Stamford, CT.

Symantec Corp. (2006), *Internet Security Threat Report*, Vol. IX [Online]
Available at: http://www.symantec.com/enterprise/threatreport/index.jsp

Thompson, H. H. (2003), "Why security testing is hard", *IEEE Security & Privacy*, 1(4), pp. 83- 86.

Viega, J., McGraw, G. (2001), *Building Secure Software: How to Avoid Security Problems the Right Way,* Addison-Wesley, Reading, MA.

Viega, J. (2004), Open Source Security: Still a Myth [Online]
Available at:
http://www.onlamp.com/pub/a/security/2004/09/16/open_source_security_myths.html

Wagner, D. (2006), "The Future of Software Security", Berkeley EECS Annual Research Symposium, February 23, 2006 [Online]
Available at: http://www.cs.berkeley.edu/~daw/talks/BEARS06.ppt

Whittaker, J. A., Atkin, S. (2002), "Software Engineering is Not Enough", *IEEE Software*, 19(4), pp. 108-115.

Whitten, A., Tygar, J. D. (1999), 'Why Johnny Can't Encrypt – A Usability Evaluation of PGP 5.0' in *Security and Usability: Designing Secure Systems That People Can Use*, ed. L. F. Cranor and S. Garfinkel, O'Reilly, Sebastopol, CA.

Williams, A.T., MacDonald, N. (2006), "Integrate Security Best Practices and Tools into Software Development Life Cycle", Gartner Inc., Stamford, CT.

Zipkin, D.S. (2005), "Using STAMP to Understand Recent Increases in Malicious Software Activity", S. M. Thesis, Technology and Policy Program, Massachusetts Institute of Technology, Cambridge, MA.

Appendix A

Threat Modeling (Howard and LeBlanc, 2002)

Thread Modeling is a structured method Microsoft has used to evaluate threats to the system. It consists of the following steps:

- Decompose the system (e.g. according to the system level data flow diagrams);
- Identify and categorize the threats using the STRIDE model (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege);
- Analyze the threats by constructing attack trees;
- Rank the threats using the DREAD categories (Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability);
- Choose the appropriate threat mitigation strategies from the corresponding STRIDE categories.

Thread modeling are used for risk analysis, design, implement, and test countermeasures to attacks (McGraw, 2006).

Attack Trees (Schneier, 2000)

Attack tree is based partially on the concept of "fault tree" in System Safety (Leveson, 1995). It models the common steps that an attack may take place (figure 10).

The root node of the tree represents the ultimate goal of the attack (for example, to view confidential payroll data on the wire, as in figure 10). The sub-nodes and leaf nodes in the tree represent the prerequisites of the attack. Each path from the leaf to the root represents a possible attack. An attack is successful only if it meets all the requirements on its path to the root node (NSCP, 2004).

Attack trees are used for risk analysis, design, implement, and test countermeasures to attacks (McGraw, 2006).

.

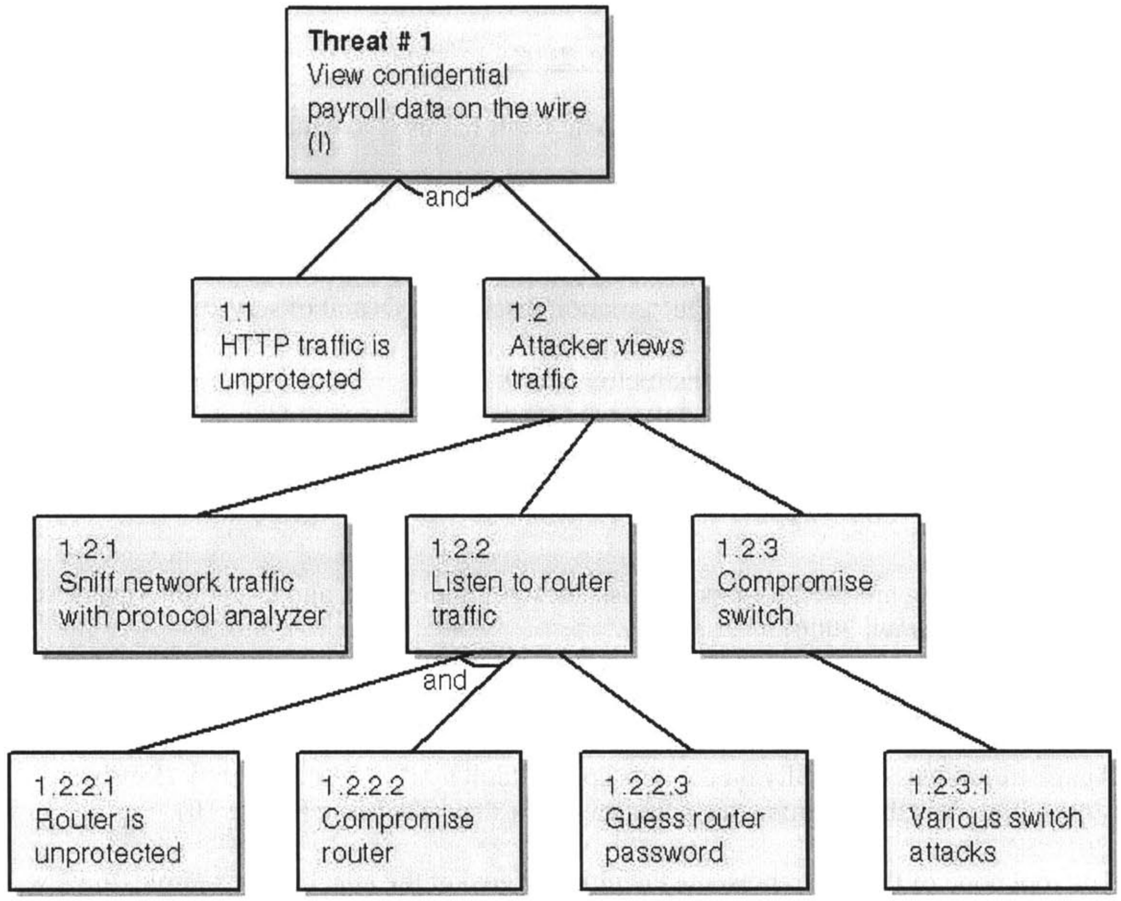


Figure 15 Sample Attack Tree *(Source: Howard and LeBlanc, 2002)*

Attack Pattern (Moore et al, 2001)

Moore, Ellison, and Linger use the term *attack pattern* to describe a common security attack. An attack pattern consists of:

- *Goal*: the purpose of the attack
- *Precondition*: the prerequisite of the attack
- *Attack*: the steps for carrying out the attack
- *Postcondition*: the consequence of the attack

In their paper (Moore et al, 2001), they defined the attack pattern for Buffer Overflow as the following:

Buffer Overflow Attack Pattern:

Goal:

Exploit buffer overflow vulnerability to perform malicious function on target system.

Precondition:

Attacker can execute certain programs on target system.

Attack:

1. Identify executable program on target system susceptible to buffer overflow vulnerability, and
2. Identify code that will perform malicious function when it executes with program's privilege, and
3. Construct input value that will force code to be in program's address space, and
4. Execute program in a way that makes it jump to address at which code resides

Postcondition:

Target system performs malicious function

Attack patterns are used for risk analysis, design, implement, and test countermeasures to attacks (McGraw, 2006).

Appendix B

Interviewees

Below is the list of people I've interviewed either through email (Apache and MSFT) and phone (MSFT).

- Steven B. Lipner, Senior Director of Security Engineering Strategy, Security Technology Unit, Microsoft Corporation
- Bilal Alam, IIS 6.0 Development Lead, Windows Division, Microsoft Corporation
- Jiri Richter, IIS 6.0 Security Tester, Windows Division, Microsoft Corporation
- Michael Howard, Senior Program Manager, SWI, Microsoft Corporation
- Paul Oka, co-chair of University Relations at Microsoft Research, Microsoft Corporation
- Yoav Shapira, member of Apache Software Foundation, PMC member of Apache Tomcat