Principle-driven Security Management

Teresa Holladay

September 2, 2003

GIAC Security Essentials Certification (GSEC)

Practical Assignment Version 1.4b
Option 1

Abstract

This paper outlines ten principles that drive security management. Principles such as "Secure the weakest link", "Practice defense in depth", and "Fail securely" seem to be make common sense and one might expect to employ them in every project. However, with the pressure of development and the limitations of budget, it is easy to fall into "checklist-driven security management", i.e., managing security based on vendor checklists. Principle-driven security management identifies issues that may never be uncovered through a vendor checklist.

While these principles were originally compiled by John Viega and Gary McGraw in their book, Building Secure Software: How to Avoid Security Problems the Right Way (Viega, pp. 90-113), they are expanded in this paper to illustrate their use across all aspects of a project, from network and system administration to web development. A different type of checklist is presented, one that provokes thought and assists the reader in understanding how to apply each principle. The benefit is that each member of the project team can evaluate their work according to these principles, thus increasing the team's ability to deliver more secure applications.

Introduction

There is an old adage that is finding a home in Information Security: "Teach them correct principles and let them govern themselves."

As information technology reaches into more areas and becomes more complex, it is no longer safe to leave security solely in the hands of administrators and security personnel. It is no longer sufficient to operate solely from "best practice" checklists provided by the vendor. In an age where malicious attackers seemingly spend 24 hours a day devising new ways to wreak havoc on electronic systems, it is imperative that each individual associated with a software project think of security every time they work on a task. Network operations, testers, project managers, and developers should all be tasked with identifying and mitigating security risks.

This can best be accomplished through Principle-driven Security Management. Principles move beyond the "how" and "what" of a checklist to examine the "why". It is the "why" that will protect an application when a weakness is not covered by a checklist.

The following principles, outlined in the book Building Secure Software: How to Avoid Security Problems the Right Way (Viega, pp. 90-113), are presented from the perspective that they can be utilized throughout an entire project, by every participant, from the manager to the developer and from the network to the Internet.

1.Secure the weakest link.
2.Practice defense in depth.

3.Fail securely.
4.Follow the principle of least privilege.
5.Compartmentalize.
6.Keep it simple.
7.Promote privacy.
8.Remember that hiding secrets is hard.
9.Be reluctant to trust.
10.Use your community resources.

Principle 1: Secure the Weakest Link

Security practitioners often point out that security is a chain. And just as a chain is only as strong as the weakest link, a software system is only as secure as its weakest component. (Viega, p. 93).

The most overlooked security principle could easily be "Secure the Weakest Link". It requires a significant investment on the part of managers and the development team to thoroughly comb through a system to uncover its weak spots.

And yet, this exercise can be the most effective in securing the system. It is well known that "attackers are opportunistic, take the easiest and most convenient route, and exploit the best-known flaws..." (SANS/FBI Top 20 List). Identifying weaknesses before the enemy does makes his job that much harder, giving the other defensive layers a chance to stop him before he successfully compromises the system..

An assessment of weaknesses is not the same as a Risk Assessment or a Vulnerability Assessment. The Weakness Assessment begins much sooner and takes place at regular intervals throughout the project. It evaluates non-technical weaknesses as well as code and technology weaknesses.

First, identify each component or process from beginning to end, covering full life cycle development. For a web application allowing access to sensitive data, such as bank accounts or health care records, this initial work list could look like this:

Components and Processes
Developers
Development Tools
Development Environment
Production Environment
Quality Assurance Process
Use of the Application; Features
System Administration
Data Administration
User Management
Help Desk Management

Next, within each category, ask, “What are our weaknesses in this area?” Continue with more probing questions. This often results in a surprising catalog of vulnerabilities in unforeseen areas. For example:

Developers
Reviewing various administration options on a software tool, are there areas with which the developer is unfamiliar?
In what areas is the developer proceeding without training or regular skills review?
Has any developer or contractor given papers or presentations describing architecture or security? Do those papers expose information about this particular application?
What are schedule constraints and issues?
What are Budget constraints and issues?

Help Desk Management
What is the process to reclaim forgotten passwords?
Do the administrative tools that allow Help Desk staff to reset passwords also give them privileges into the various client products?
What security training do Help Desk personnel receive?
Do Help Desk personnel participate in “social engineering” tests to assess vulnerability in this area?
Does Help Desk administration of userids and passwords on a secure application offer a “single point of failure”?

A comprehensive catalog of weaknesses can then become a roadmap to a security improvement plan that includes training on vendor products, purchase of additional tools, and a re-design of Help Desk procedures. None of these solutions would have been covered by a vendor security checklist.

Principle 2: Practice Defense in Depth

Have a series of defenses so that if an error isn't caught by one, it will probably be caught by another. (MacLennan, p. 526).

Originally a military concept, “defense in depth” utilizes a variety of checkpoints to ensure that there are many opportunities to stop an attacker before he gets to the prize.

The first checkpoint presumes that the facility has identified the adversaries who might try to breach the perimeter. For example, the security at a local military facility might primarily be designed to keep out a curious but non-threatening public. However, the security at a command center in a war zone is intended to keep out a hostile, determined, and well-armed enemy. Subsequent checkpoints are designed to thwart the advance of anyone who makes it past the first checkpoint.

> "Defense in depth increases security by raising the cost of an attack. This system places multiple barriers between an attacker and your business-critical information resources: The deeper an attacker tries to go, the harder it gets. These multiple layers prevent direct attacks against important systems and avert easy reconnaissance of your networks. In addition, a defense-in-depth strategy provides natural areas for the implementation of intrusion-detection technologies. Ideally, the defense-in-depth measures you implement should buy you time to detect and respond to a breach, reducing its impact." (Carr, 2001)

It is a common practice in security administration to jump right to solutions. A web administrator might enable SSL, ensure authentication and authorization protocols, configure firewalls, and install a host-based intrusion detection system. However, if he has not identified his potential adversaries, he has missed the first step of defense in depth and possibly missed additional opportunities to secure his system.

The National Security Agency's white paper on Defense in Depth (NSA, 2001) recommends beginning with a careful examination of potential adversaries, their motivations, and their classes of attack.

Who are potential adversaries of this system?

- Nation States
- Terrorists
- Criminal Elements
- Hackers
- Corporate Competitors
- Disgruntled employees
- Disaffected beneficiaries or users
- Other: ___________________

What are their motivations?

- Intelligence gathering
- Theft of intellectual property
- Identity theft
- Terrorism (in theft of personal identifying information)
- Denial of service
- Theft of service (use of the servers or other equipment)
- Embarrassment
- Revenge
- Pride in exploiting a notable target
- Other: ___________________

What are their classes of attack?

- Passive monitoring of communications
- Active network attacks
- Close-in attacks
- Exploitation of insiders (social engineering)
- Attacks through the providers of the IT network
- Physical theft of hard drives containing sensitive information
- Session hijacking (Note: Sophisticated but difficult to defend against within the application)
- ATM Attacks, “phishing” (a type of spoofing attack)
- Other: ___________________

Once you identify attackers, their motivations and common means of attacking an application, you can organize an effective “Defense in Depth” strategy.

Principle 3: Fail Securely

Failure is unavoidable and should be planned for. What is avoidable are security problems related to failure. (Viega, p. 97).

Credit card authentication is the best real-world example of the potential for insecure failure. Swiping a card magnetically provides instant access to spending limits and can alert a merchant to a stolen card. Since magnetic strips can be copied, card issuers include codes on the back of the card that must be keyed at time of sale. A significant point of failure occurs when the magnetic strip on the back of the card fails. Merchants must create a manual imprint of the card. This method bypasses many of the issuer's authentication systems, increasing the risk of fraud. Issuers now mitigate this point of failure by providing telephone validation and authorization codes for manual transactions. (Burns, 2002).

Buffer overflows are the best example of insecure failure in the software industry. Buffer overflows occur when programs do not adequately check input for appropriate length. Unexpected input “overflows” into the CPU execution stack. Exploited by a knowledgeable and malicious programmer, this overflow can be used to launch code of the programmer's choice. (Scambray, p.160).

Risk assessment should identify key checkpoints and answer the question: What happens if this checkpoint fails?

For example, a Web Portal desires to pass credentials (single sign-on) to a downstream application hosted on a different server. Risk assessment identifies the following checkpoints in context of specific proposed techniques and examines what would happen if the checkpoint failed.

- Referring site validation
- Routers
- Firewalls
- Host-based Intrusion Detection System
- User authentication
- Privilege authorization
- Limitation on privileges (i.e., select only; read directories; upload files)
- Server and/or client certificates
- Data-in-transit encryption
- Data-at-rest encryption
- Database authentication
- Logging
- Help Desk

Listing each checkpoint and evaluating the consequences of failure at each step can ensure that the application is able to fail securely at one point without exposing the entire application or environment to further risk.

Principle 4: Follow the Principle of Least Privilege

The principle of least privilege states that only the minimum access necessary to perform an operation should be granted, and that access should be granted only for the minimum amount of time necessary. (Viega, p. 100).

The Principle of Least Privilege is associated with a term that is more familiar to the public: Need-to-know. "Need-to-know" is the practice of granting security clearance only to those who "need to know" certain information.

The Department of Defense defines Need-to-know as follows: **"**A determination made by an authorized holder of classified information that a prospective recipient requires access to specific classified information in order to perform or assist in a lawful and authorized governmental function". (DOD 5200.1-R).

The Principle of Least Privilege, or Need-to-know, limits the damage that can be done by a trusted insider who goes bad. It can also limit the damage that can be done when one component or feature fails.

In software development, this principle can best be illustrated through the use of a particular Microsoft Security Template. Windows NT was known to be less secure than Windows 2000. However, a number of desktop applications were designed to run with expansive privileges within the Registry. Microsoft included a security template, compatws.inf, which "undid" the higher levels of security afforded by Windows 2000 and "rolled back" security to Windows NT levels. In many cases, this granted the "Everyone" group "full control" privileges to the applications.

Microsoft was aware that some administrators might not understand the implications. Instructions regarding the use of these templates included the disclaimer: "The Compatible template opens up the default permissions for the Local Users group so that legacy programs are more likely to run. This configuration is not considered a secure environment." (Microsoft Knowledge Base article 234926).

In order to define what might be considered a "least privilege", it is helpful to assess what privileges are granted. In the following example, users are accessing a secure extranet:

Category	**Privileges**
Login	▪ Users cannot login without valid userid and password ▪ Users must come from a specific domain or portal ▪ Userids issued only to users with mycompany.com email address
Applications	▪ Users will only see links to applications for which they are authorized access
Privileges	▪ "Advanced" users are permitted to download extracts to their workstation
Time-out	▪ Idle users will timeout within 15 minutes ▪ Userids with no activity within 30 days will be deactivated ▪ Userids with not activity within 90 days will be terminated

An inventory of privileges is the first step towards identifying and restricting "least privilege.

Principle 5: Compartmentalize

The basic idea behind compartmentalization is to minimize the amount of damage that can be done to a system by breaking up the system into as few units as possible while still isolating code that has security privileges. (Viega, p. 102).

A submarine is built so that if a breach in the hull causes flooding in one chamber, it can be sealed off to prevent flooding in another chamber.

In software design, when one part of an application is breached, it is "sealed off" from other parts to prevent further intrusion.

This concept is closely related to Principle 3: Fail Securely. At the Operating System level, it is difficult to achieve. For example, in the standard UNIX privilege model, operations work on an all-or-nothing basis. A user with root privileges is permitted to access any application on the system.

However, developers can isolate databases, database privileges, applications, application privileges, and user roles. Compartmentalization should not be so unwieldy as to make the application unusable.

Create a worksheet that lists security components. How are they isolated from each other? If they are not isolated, what are the benefits and risks? It is possible that asking the questions raises issues that need further exploration before they can be deemed acceptable.

Continuing the example used above, examine components of a secure extranet:

Component	**Benefits**	**Risks**
LDAP authentication: Shared between Business Intelligence Tool and Database	Single resource for both login and statistical reporting.	If front-end tool is breached, attacker has access to database
Time-out: User interface has different time-out (15 minutes) from database (2 hours)	N/a	Why is the database time-out different from the user interface? (To facilitate behind-the-scenes datamart population). Can the database time-out be set by user or task?

Note that simply creating the inventory exposes weaknesses and questions that might not otherwise have been discovered. The net effect is to create a security-aware development team that identifies potential problems early in the project, thus providing adequate time to devise a solution.

Principle 6: Keep it Simple

Complexity increases the risk of problems. Avoid complexity and avoid problems. (Viega, p. 104).

This principle seems at odds with Defense-in-Depth principles. As with the Principles of Fail Securely and Compartmentalize, the key is to strike the right balance between functionality, maintenance, and security.

How does one achieve simplicity in a complex network or application? Through code and component re-use and chokepoints (funneling operations through a small number of controllable interfaces).

Identify what aspects of code can be re-used:

- Error management
- User authentication
- Role-based privileges
- Statistical reporting

List chokepoints:

- Error identification components
- Logging
- Navigation

Principle 7: Promote Privacy

Promote privacy for your users, for your systems, and for your code. (Viega, p. 109).

The average Internet consumer may understand that "privacy" to mean one of the following:

1. Personal user information, such as name or email address, is not shared with 3rd parties.
2. Personal financial information, such as bank accounts or credit card numbers, either is encrypted in a database or is not stored at all.
3. "Cookies" or other means of tracking user activity are not used or are used only with the user's permission.

For the developer and System Administrator, however, "privacy" also includes protecting the code, the system, and the network. Servers, web application software, coding techniques and even unique "quirks" can inadvertently reveal information to an attacker.

For example, if a hacker observes the use of ".cfm?fuseaction=", he knows that this is a ColdFusion website using the Fusebox coding technique. The presentation of the variables after the "=" tells him which version of Fusebox is being used. Since attackers generally prey on easy targets, knowing this information does not necessarily open doors for him, however, it makes his next steps easier. Knowing typical ColdFusion exploits, he can scout around a little further to probe for common vulnerabilities.

Each platform or technique offers ways to hide revealing information. In some cases, it may even be prudent to substitute good information with misleading information; it would buy system administrators time to identify probes and block an attempted attack.

To help protect the privacy of both the users and the system, itself, compile a list of all of the basic components. Identify what is revealed. Assess the vulnerabilities and risk if that information is exposed. Examine various ways where that information can be masked or made to mislead.

Such a list might look like this:

Components	Info Revealed	Vulnerabilities	Mitigation
User privacy			
Personal identifying information	User name and address are presented on an validation page.	Can be viewed in transit (high risk). Printed out by user (low risk).	Enable SSL.
Login info	Userid is displayed at the top of each web page.	Can be viewed in transit (high risk). Appears on all printouts.	Enable SSL. Remove userid from the top of each page.
Social Security Number	SSN is used to set up userids but is not captured in any database.	Paper applications can be misplaced, stolen, or improperly used by insiders.	Alert client to risks associated with paper filing, ensure adequate handling.
Credit Card info	Not captured.	N/a	N/a
Server privacy			
Database server OS	Telnet reveals Unix OS, version.	Attacker can attempt common exploits (high).	Ensure that common vulnerabilities are patched or configured.
Database	Error page reveals SQL Server.	Attacker can attempt common exploits (high).	Ensure that common vulnerabilities are patched or configured.
Web server	Telnet reveals Apache server, version. Error page reveals directory structure.	Attacker can attempt common exploits (high).	Ensure that common vulnerabilities are patched or configured. Apply customized error pages.

Keep in mind that the intent is to slow down an attacker so that another defensive layer can stop him.

Principle 8: Remember that Hiding Secrets is Hard

Software is a powerful tool, both for good and evil. Because most people treat software as magic and never actually look at its inner workings, the potential for serious misuse and abuse is a very real risk. (Viega, p. 111).

This principle applies primarily to the most likely attacker: The insider. Binaries can be reverse-engineered, software keys and algorithms can be revealed, private user information can be accessed and abused.

It goes against the grain of a well-run team to presume that anyone could abuse the “secrets” hidden within the data or the application, yet compromises by insiders continue to make the press. An estimated 80% of all computer and Internet-related crime is committed by insiders (Network Magazine, 2002). FBI spy Richard Hanssen carried out his insider attacks for more than 15 years. Lawsuits among software vendors claim theft of intellectual property, presumably from inside sources (Haley, 2002).

It would be prudent to identify the types of “secrets” that are present within the system, identify possible adversaries as noted in Principle 3: Fail Securely, evaluate risk, and mitigate accordingly.

Principle 9: Be Reluctant to Trust

When you spread trust, be careful. (Viega, p. 112)

Trust is a tool that should be used like fire: Banked and contained, it is a powerful tool, but spread indiscriminately, it becomes an overpowering foe.

Kevin Mitnick, the hacker famous for his social engineering exploits, claims that he never actually asked for a password. Instead, he says that he built relationships and trust with everyone from System Administrators to Help Desk personnel. “That is the whole idea,” he said, “to create a sense of trust and then exploiting it” (ZDNet, 2000).

Trust relationships extend beyond individuals. There are trust relationships between servers, software hosted on the server (such as an SMTP mail server), and databases. There is also the trust relationship between a vendor and a client.

In the writer's own experience, a significant vulnerability was found in a software patch provided by the vendor. Because the team questioned everything, a Registry Monitor tool was engaged during testing. The vendor had indicated that a patch was needed in order to run that software under Windows 2000. The Registry Monitor revealed that the patch gave the Everyone group Full Control at the Registry's root.

This exposes the Registry to unauthenticated users, allows viruses to be installed, and negates Company security measures. The vendor was tasked with supplying a new patch that granted permissions at a much more restricted level.
Be reluctant to trust any product, any vendor, or any individual. Think twice about activities as benign as applying a patch or establishing a trust relationship between two servers. Ensure that measures are in place to monitor the action and to mitigate any consequences.

Principle 10: Use Your Community Resources

There is something to be said for strength in numbers. Repeated use without failure promotes trust. (Viega, p. 112).

The great paradox of security management is that while one principle states that security personnel should be reluctant to trust, this final principle encourages just the opposite. The key is to know who to trust and when. In the software industry, it is possible that “experts” and even the public can be good judges of character, at least as far as software or coding technique can go.

Summary:

Security management by principle requires significant thought and investment. Solutions to issues cannot be readily found in one, comprehensive checklist or source. They are unique to each system or application.

The primary benefit of Principle-driven security management is that it expands the security team to include every developer, every tester, every administrator, and every manager. It promotes a more thorough evaluation of risks and consequences and results in a greater ability to secure the product.

References:

Haley, Colin C. "Settlement Suits Cognos, Business Objects". Boston Internet.com. May 28, 2002.
URL: http://boston.internet.com/news/article.php/1145131.

Burns, Peter and Stanley, Anne. *Fraud Management in the Credit Card Industry.* Federal Reserve Bank of Philadelphia, Payment Cards Center White Paper. April 2002.
URL: http://www.phil.frb.org/pcc/workshops/workshop7.pdf

Department of Defense Information Security Program. DOD 5200.1-R
URL: http://www.defenselink.mil/nii/other/reg52001.html

Lima, Joe. "Mask Your Web Server for Enhanced Security". Port80 Software.
URL: http://www.port80software.com/support/articles/maskyourwebserver

MacLennan, Bruce. Principles of Programming Languages. New York: Holt, Rinehart and Winston, 1987. p. 526.

Microsoft Knowledge Base Article – 234926
"Windows 2000 Security Templates are Incremental". May 14, 2003.
URL: http://support.microsoft.com/?kbid=234926

National Security Agency. "*Defense in Depth".* 2001.
URL: http://nsa1.www.conxion.com/support/guides/sd-1.pdf

Paul, Brooke. "Building an In-depth Defense". Network Computing. July 9, 2001.
URL: http://www.networkcomputing.com/1214/1214ws1.html

Carr, Jim. "Strategies & Issues: Thwarting Insider Attackers". Network Magazine. Sept. 4, 2002.
URL: http://www.networkmagazine.com/article/NMG20020826S0011

SANS/FBI Top 20 List: The Twenty Most Critical Internet Security Vulnerabilities Version 3.23, May 29, 2003. URL: http://www.sans.org/top20/

Scambray, Joel, McClure, Stuart, and Kurtz, George. Hacking Exposed: Network Security Secrets and Solutions. Second Edition. Berkeley: Osborne McGraw-Hill, 2001. p. 160.

Viega, John and McGraw, Gary. Building Secure Software: How to Avoid Security Problems the Right Way. Addison-Wesley. Indianapolis, 2002. pp. 90-113.

ZDNet News. "Mitnick teaches 'Social Engineering'". July 16, 2000.
URL: http://zdnet.com.com/2100-11-522261.html?legacy=zdnn