Two security patterns: Least Privilege and Security Logger and Auditor

Eduardo B. Fernandez[1], Sergio Mujica[2], and Francisca Valenzuela[2]

1 Dept. of Comp. Science and Eng., Florida Atlantic University, Boca Raton, FL, USA
ed@cse.fau.edu
2 Escuela de Informatica, Universidad Andres Bello, Santiago, Chile, smujica@unab.cl

Abstract
We present here two security patterns that describe fundamental aspects: ***Least Privilege--*** How can we minimize misuses by the users or workers of an institution? Give the users or the executing processes of a system the rights they need to perform their functions and no more. ***Security Logger/Auditor--*** How can we keep track of user's actions in order to determine who did what and when? Log all security-sensitive actions performed by users and provide controlled access to records for Audit purposes.

Introduction
We present here two patterns that describe fundamental security aspects:

Least Privilege-- How can we minimize misuses by the users or workers of an institution? Give the users or the executing processes of a system the rights they need to perform their functions and no more.

Security Logger/Auditor--
Log all security-sensitive actions performed by users and provide controlled access to records for Audit purposes.

Least privilege is a fundamental security principle. We intend to write patterns for all the principles in [Sal75].

Logging is one of the three basic security mechanisms, that should be present in any secure system, the others being Authentication and Authorization/Access Control.

Least privilege
Intent
How can we minimize misuses by the users or workers of an institution? Give the users or the executing processes of a system the rights they need to perform their functions and no more.

AKA
Need-to-know, Principle of Least Authority (POLA)

Example
A hospital uses Role-Based Access Control (RBAC) to define the rights of its employees. For example, doctors and nurses can read and write medical records and related patient information (lab tests, medicines,…). When a famous patient came to the hospital, one of

the doctors, who was not treating him, read his medical record and leaked this information to the press. In another incident, a malicious employee changed the amount of medication of a patient with the intent of harming her.

Context
Any system where users must be given rights to use resources (objects).

Problem
When users are given too many rights, the probability that they will abuse them will increase. How can we minimize misuses by the users or workers of an institution?

The solution to this problem is driven by the following **forces**:

- ***Misuse avoidance***. If users or processes are given too many rights, it is possible for them to perform misuses of the resources to which they have access. An executing process cannot attack other processes or access unauthorized data if its rights are limited.

- ***Default user rights***. There should be no other rights except those explicitly given. Otherwise, we cannot know if we gave only the needed rights.

- ***Authorization*.** There must be a way to give rights to users so that resources (objects) should only be accessed by authorized users. These rights should have an appropriate granularity.

- ***Authentication***. If we do not know who is in the system, we cannot enforce any rights given to users or processes.

- ***Error avoidance***. Restricting rights decreases the effects of accidental errors that now can compromise a smaller number of items.

- ***Performance.*** Giving too fine rights may result in performance overhead.

- ***Administration***. Giving too fine rights may result on extra work for administrators and may lead to their confusion.

- ***Productivity.*** A restrictive use of this policy may hinder the productivity of users or applications.

Solution
Give the users or the executing processes of a system the rights they need to perform their functions and no more. This implies a c*losed system,* where no authorization means no access and having an access control system able to enforce a fine granularity access. It also implies a security administrator role, that can grant, revoke, or modify rights.

Implementation

Since least privilege is a policy or a design principle, it can be applied in a variety of ways. We discuss some possibilities below. In fact, these can be patterns in their own right.

To apply this principle to users, determine a minimum set of rights from the system use cases: For each user, see in which use cases she participates and give her only the rights needed to perform her duties. [Fer97] shows that for Role-Based Access Control (RBAC), we can determine the need-to-know rights by analyzing all the use cases of the system.

Rights assignment

Structure

Figure 1 illustrates the application of the pattern. The **Authorizer** creates **Rights** and assigns them to **Subjects** in order to let them access **Objects** (resources). The Right includes an access type and may include a predicate for content-dependent access control. To do the assignment, the Authorizer checks the predefined Rights associated with the U**ser functions** and assigns them to the users (**Object Rights**). The rights are only the minimal rights they need to perform their functions. Some **Users** can act as Authorizers, i.e. they can give rights to other users if they have administrative functions. Users then become subjects for authorization rights.

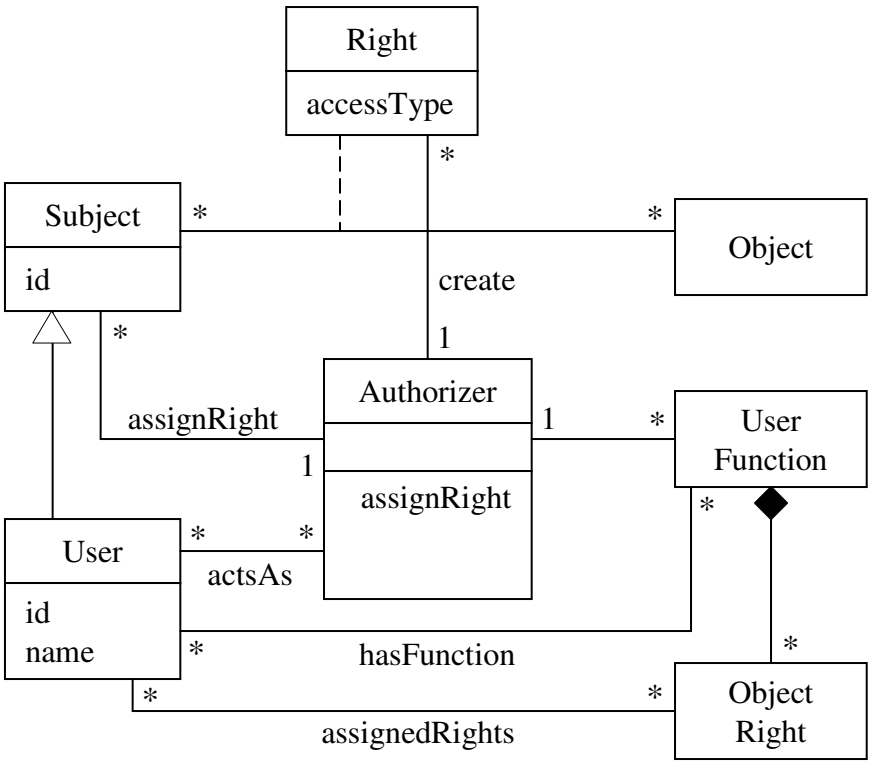

Figure 1. Class diagram of the Need-to-Know Authorizer pattern

Dynamics

Figure 2 shows a sequence diagram for the use case Assign Rights to a User. A user requests rights from an Authorizer (security administrator role), who checks his Function Rights (Object rights derived from his Function), and creates the corresponding rights. The rights are then assigned to the User.

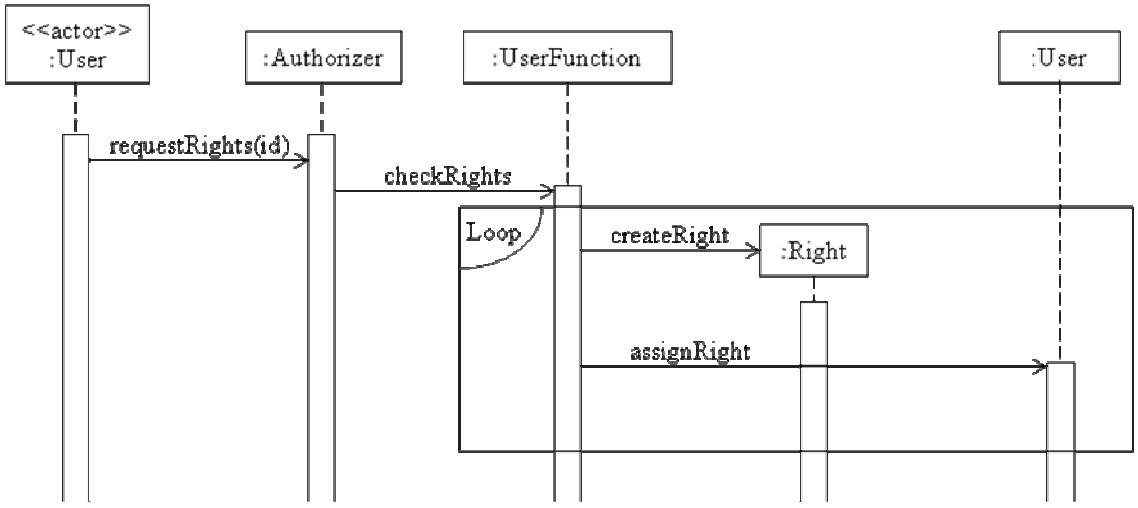

Figure 2. Sequence diagram for use case Assign Rights to a User

Enforcement

Enforcement depends on which architectural level we are applying this principle. For example, in a DBMS we can enforce this principle using views. Figure 3 shows a possible enforcement structure. **Users** are given **DataViews** that are collections of **Rights**, including access types (maybe restricted by predicates) to access (read, modify) **DataItems** in a DBMS.

For processes, enforcement can be done using Protection Rings [Fer08] or using a Controlled Virtual Address Space (Sandbox) [Sch06].

Other approaches to enforcement are:

- **Privileges in Batch mode**: Sometimes to reduce security risk, a system operates in various batch modes where a particular set of privileges will be granted as a unit before execution. The execution of the preapproved user requests are scheduled according to the batch execution sequence.

- **Virtual update privileges**: Some data updates (e.g. wikipedia updates) need to be double-checked before making the updates persistent. The updates for such data can be made as virtual updates where a temporary copy of the data is made. The data updates can be made on the temporary copy, the original copy of the data is modified upon approval from a higher authority.

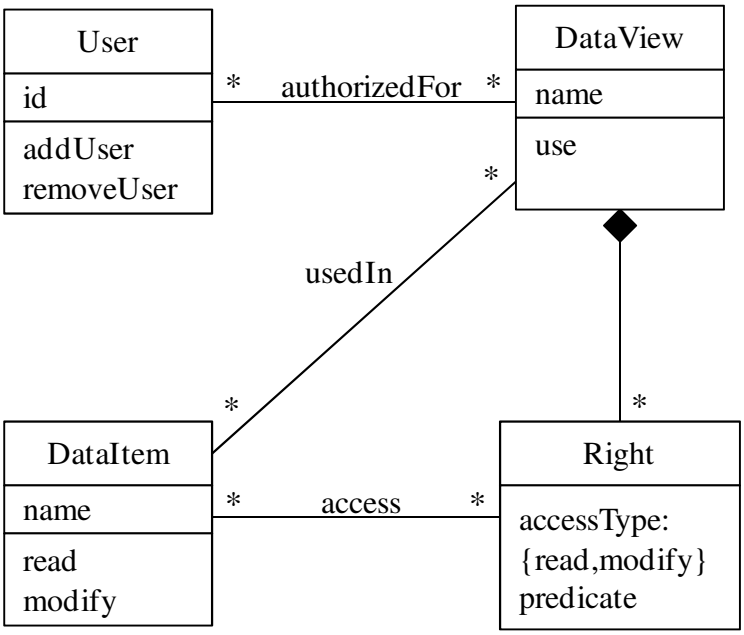

Figure 3. Enforcement using views

Known uses

- Making email address lists use authorization to control access to their contents can control virus propagation [Fer10]. Other virus actions can be controlled similarly [Kar03].

- Operating systems such as Windows use this principle to control the execution of processes [MS06].

- [wik10] mentions applications in the SELinux and Solaris operating systems.

Example resolved

After the incident, the hospital started applying a need-to-know policy for accessing patient records. Now doctors can read and modify only the records of their own patients. This approach can prevent most cases like the one above.

Consequences

This pattern has the following ***advantages***:

- ***Closed system***. Omitting a right does not produce a potential security violation.

- ***Authorization*.** Every access is mediated and we can apply any degree of fineness to rights.

- ***Misuse avoidance***. Users can now perform accesses only to resources for which they have rights.

- ***Forensics and auditing***. If a misuse has happened, it is easier to determine who did it since users have more precise rights.

- ***Error avoidance***. Since users don't have unnecessary rights they can perform less damage in case of error.

- ***Performance***. It is possible to control the fineness of rights so that there is acceptable performance overhead.

- ***Administration.*** It is possible to define policies such that there are not too many rights and they can be handled conveniently.

- ***Productivity***. Using the definition of task as a guideline we can give each user enough rights to perform her job.

The solution also has some ***liabilities***:

- Some misuses are still possible because users may misuse their legitimate rights. We can control those using policies such as Logging/Auditing and Separation of Duty
- We need to have an access control system with appropriate fineness or there will be no way to enforce the defined rights.
- We need to have a clear definition of functions so we know which rights are needed to perform each function. A method to define rights for roles is shown in [Fer97].
- A too-restrictive use of this policy could hinder the work of users or applications.

Related patterns

- Role Rights Definition [Sch06], based on [Fer97], applies this policy when defining the rights of users starting from use cases.
- The Controlled Virtual Address Space (Sandbox) pattern defines a structure to apply this principle to constrain the execution of processes [Sch06].
- The Protection Rings pattern provides architectural support to apply least privilege to processes [Fer08].
- The Reference Monitor [Sc06], is an abstract pattern that describes the way to perform enforcement.

Security Logger and Auditor

Intent
How can we keep track of user's actions in order to determine who did what and when? Log all security-sensitive actions performed by users and provide controlled access to records for Audit purposes.

AKA
Audit Trail

Example
A hospital uses RBAC to define the rights of its employees. For example, doctors and nurses can read and write medical records and related patient information (lab tests and medicines). When a famous patient came to the hospital, one of the doctors, who was not treating him, read his medical record and leaked this information to the press. When the leak was discovered there was no way to find out who was the doctor that had accessed the patient's records.

Context
Any system that handles sensitive data, in which it is necessary to keep a record of access to data.

Problem
How can we keep track of user's actions in order to determine who did what and when? The solution to this problem is driven by the following **forces**:

- ***Accuracy.*** We should faithfully record what a user or process has done with respect to the use of system resources.

- ***Security***: Any information we use to keep track of what the users have done must be protected. Unauthorized reading may reveal sensitive information. Tampering may erase past actions.

- ***Forensics:*** When a misuse of data occurs it may be necessary to audit the access operations performed by users to determine possible unauthorized actions and maybe trace the attacker or understand how the attack occurred.

- ***System improvement***. The same misuses may keep occurring; we need to learn from past attacks.

- ***Compliance.*** We need a way to verify and to prove to third parties that we have complied with institution policies and external regulations.

- ***Performance.*** We need to minimize the overhead of logging.

Solution

Each time a user accesses some object, we record this access, indicating the user identifier, the type of access, the object accessed, and the time when the access happened. The database of entries must have authentication and authorization systems and maybe an encryption capability.

Structure

In Figure 4, **User** operations are logged by the **LoggerAuditor**. The Logger Auditor keeps the **Log** of user accesses, where each access is described by a **LogEntry**. The Security Administrator (**SecAdmin**) activates or deactivates the Log. The **Auditor** can read the log to detect possible unauthorized actions.

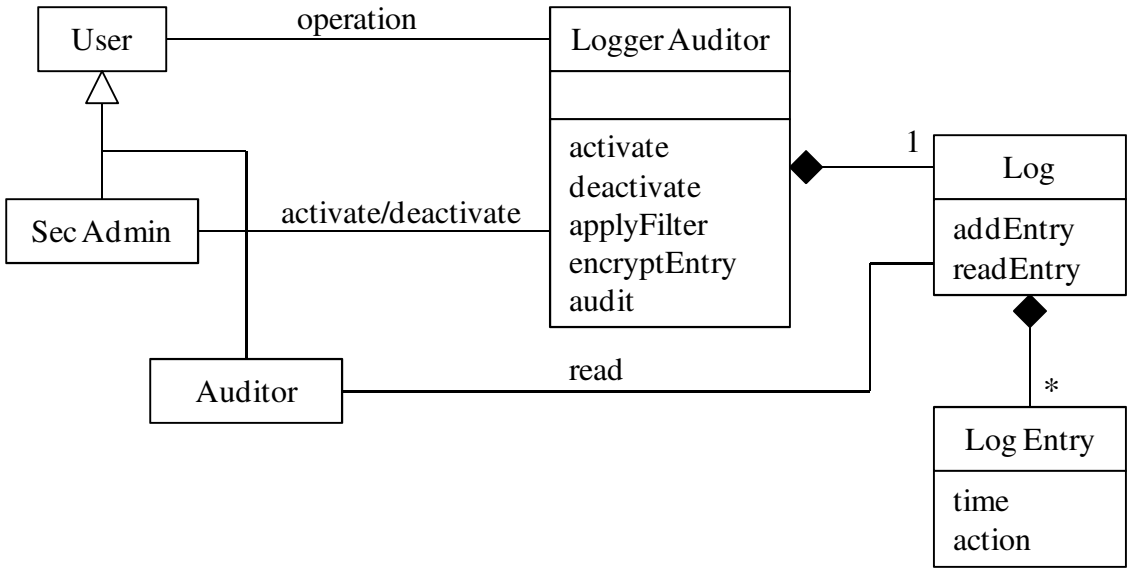

Figure 4. Class diagram of the Secure Logger/Auditor pattern

Dynamics

Possible use cases include: Log User Access, Audit Log, Query LogDB

A sequence diagram for the use case Log User Access is shown in Figure 5. The user performs an operation to apply an access type on some object: operation (accesstype t, object o). The Logger adds an entry with this information in the Log and the name of the user. The Log creates an entry adding also the time of the operation.

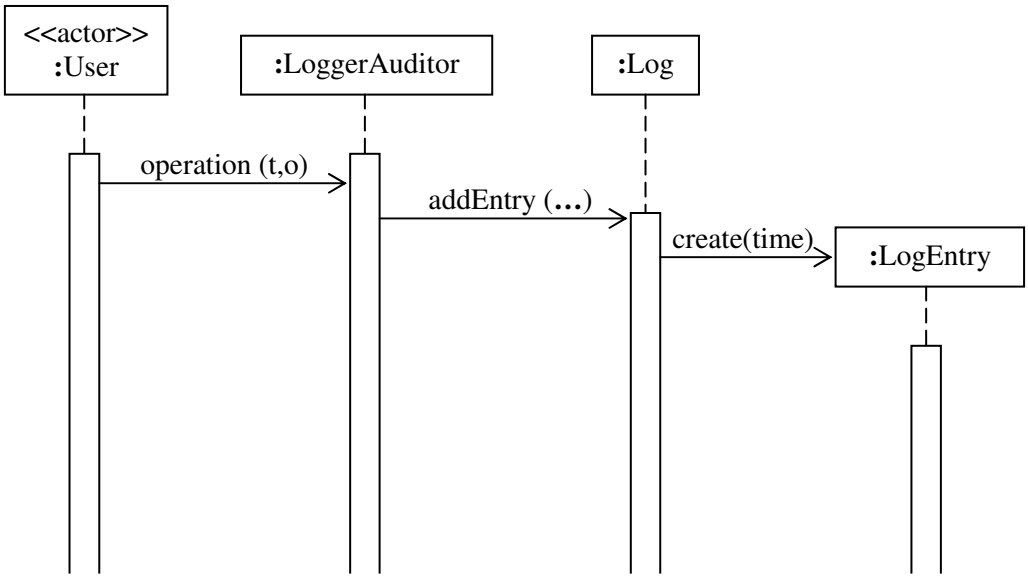

Figure 5. Use case Log User Access

Implementation
The class diagram of Figure 4 provides a clear guideline for implementation since its classes can be directly implemented in any object-oriented language. We need to define commands to activate or deactivate logging, apply filters, indicate devices to be used, allocate amount of storage, select security mechanisms. One can filter some logging by selecting users, events, importance of events, times, and objects in the filters. Administrative sec actions, e.g. account creation/deletion, assignment of rights, and others, must also be logged.

Logging is performed by calling methods on the Logger class. Every non-filtered user operation should be logged. Logged messages can have levels of importance associated with them.

Audit is performed reading the log by an auditor. This can be complemented with manual assessments that include interviewing staff, performing security vulnerability scans, reviewing application and operating system access controls, and analyzing physical access to the systems [sau]. The MVC pattern can be used to visualize the data using different views during some complex statistical analysis of the log data.

Example resolved
After the incident, the hospital installed a Security Logger so in the future such violations could be discovered.

Variants
Most systems have a System Logger, used to undo/rollback actions after a system crash. That type of Logger has different requirements but sometimes is merged with the Security Logger [SAP01]. System logs are of interest to system and database

administrators, while security logs are used by security administrators, auditors, and system designers.

Another variant could include the automatic rising of alarms by periodic examination of the Log, searching records that match a number of rules that characterize known violations. For example, Intrusion Detection Systems use this variant.

We can also add logging for reliability, to detect accidental errors.

Known uses

Most modern operating systems, including Microsoft Windows [smi04], AIX Solaris [aix], and others have security loggers.

SAP uses both a security audit log and a system log [SAP01].

Consequences
This pattern presents the following **advantages:**

- ***Security***: It is possible to add appropriate security mechanisms to protect the recorded data, e.g. access control and/or encryption.

- ***Forensics:*** Enables forensic auditing of misused data objects. Records of access can be used to find if someone has maliciously gained access to data. This pattern can also be used to log access by system processes to data objects. For example, malicious code planted in the system can be tracked by finding processes that have misused objects.

- ***System improvement***. By studying how past attacks happened, we can improve the system security.

- ***Compliance.*** Auditing a log can be used to verify and prove that institutional and regulatory policies have been followed.

- ***Performance.*** We can reduce overhead by parallel or background logging. We can also not log some events not considered significant. Finally, we can merge this log with the recovery log, needed for possible rollback.

The pattern has the following **liabilities:**

- It can incur significant overhead since each object access has to be logged. See above.

- A decision must be made by software designers as to the grain size at which objects are logged. There is a tradeoff between security and performance.

- It is not easy to perform forensic analysis and specialists are required.

- Protecting the log adds some overhead and cost.

Related patterns

- The Secure Logger is a pattern for J2EE [Ste06]. It defines how to capture the application-specific events and exceptions to support security auditing. This pattern is mostly implementation oriented and does not consider the conceptual aspects discussed in our pattern. It should have been called a security logger because it does not include any mechanisms to protect the logged information.

- M. Fowler has an Audit Log analysis pattern [Fow] for tracking temporal information. The idea is that any time something significant happens you write some record indicating what happened and when it happened.

- Authentication [Sch06]. How can we make sure that a subject is who he says he is?

- Authorization [Sch06]. How can we control who can access which resources and how in a computer system?

Conclusions

These two patterns focus on fundamental aspects of security and as such they have a wide application. Least privilege is a basic principle for security that should be applied in all systems in every architectural level. Log and audit form a basic security mechanism, required in any system that handles sensitive information.

Future work includes patterns for Database Security using Views, and other principles such as Separation of Duty.

Acknowledgements

We thank our shepherd Kiran Kumar for his valuable comments that significantly improved our paper.

References

[aix10] AIX system security auditing, http://www.aixmind.com/?p=1019

[Fer97] E. B. Fernandez and J. C. Hawkins, “Determining role rights from use cases”, *Procs. 2nd. ACM Workshop on Role-Based Access Control*, November 1997, 121-125.

[Fer08] E.B.Fernandez and D. LaRed M., "Patterns for the secure and reliable execution of processes". *Procs. of the 15th Int.Conference on Pattern Languages of Programs (PLoP 2008)*, colocated with OOPSLA, Nashville, TN, Oct. 2008.

[Fer10] E.B.Fernandez, N. Yoshioka, and H. Washizaki, "A Worm misuse pattern", *Procs. of the 1st Asian Conference on Pattern Languages of Programs(AsianPLoP 2010)* , Tokyo, Japan, March 16-17, 2010, http://patterns-wg.fuka.info.waseda.ac.jp/asianplop/ (last accessed Jan. 20, 2011)

[Fow] M. Fowler, "Audit Log", http://martinfowler.com/ap2/auditLog.html

[Kar10] A.H. Karp, "Enforce POLA on processes to control viruses", *Comm. ACM*, vol. 46, No 12, December 2003, 27-29.

[MS06] Microsoft, "Applying the Principle of Least Privilege to User Accounts on Windows XP ", http://technet.microsoft.com/en-us/library/bb456992.aspx (last accessed Jan. 20, 2011).

[Sal75] J. H. Saltzer and M.D.Schroeder, "The protection of information in computer systems", *Procs. of the IEEE*, vol. 63, No 9, September 1975, 1278-1308. http://web.mit.edu/Saltzer/www/publications/protection/index.html (last accessed January 20, 2011)

[SAP01] SAP AG, Comparing rge security audit log and the system log,2001 http://www.google.com/#sclient=psy&hl=en&q=security+logger+auditor&aq=&aqi=&aql=&oq=&pbx=1&fp=c801187a8d9c3641

[sau] http://en.wikipedia.org/wiki/Information_technology_security_audit

[Sch06] M. Schumacher, E.B.Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering,* Wiley 2006.

[Smi04] Randall F. Smith Auditing users and groups with the Windows security logWindows 20041http://www.windowsecurity.com/articles/Auditing-Users-Groups-Windows-Security-Log.html

[Ste06] Chapter 9 in "Securing the Web Tier: Design Strategies and Best Practices," .

[wik] Principle of least privilege, http://en.wikipedia.org/wiki/Principle_of_least_authority (last accessed Jan. 20, 2011).