JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 1, January-February 2004

# Specifying Reusable Security Requirements

**Donald Firesmith**, Software Engineering Institute, U.S.A.

**Abstract**

Unlike typical functional requirements, security requirements can potentially be highly reusable, especially if specified as instances of reusable templates. In this column, I will discuss the concepts underlying security engineering including its quality subfactors. I will then address the issue of security requirements and how they differ from the architectural mechanisms that will fulfill them. Then, I will discuss the value of reusable parameterized templates for specifying security requirements and provide an example of such a template and its associated usage. Finally, I will outline an asset-based risk-driven analysis approach for determining the appropriate actual parameters to use when reusing such parameterized templates to specify security requirements.

## 1 CONCEPTS UNDERLYING SECURITY ENGINEERING

To specify security requirements, it is critical to first understand the concepts underlying security engineering. And the most important concept of these is 'security' itself. Whereas security is often defined as an incomplete subset of its most important quality subfactors (e.g., integrity and privacy), the following figure illustrates that a more general and complete definition of security is that it is the degree to which *malicious*[1] (i.e., unauthorized and intentional) harm to valuable system assets is prevented, reduced, and properly responded to. Thus, security is about protecting these assets (e.g., data, services, hardware, and personnel) from harm due to various kinds of attacks (e.g., password sniffing, spoofing, viruses) that may be mounted by the various kinds of attackers (e.g., hackers, crackers, disgruntled employees, international cyber-terrorists, industrial spies, governmental spies, foreign military, etc.). These assets are at risk due both to various kinds of threats (e.g., theft, vandalism, unauthorized disclosure, destruction, fraud, extortion, espionage, trespass, etc.) of attack as well as the vulnerabilities the system may

[1] Some may argue that the term 'malicious' is too strong. What about people who vandalize the website of a company that pollutes the environment? What about someone who uses company computers to surf the Web in violation of company policy. The first example is a cybercrime and the second is an unauthorized use of property. In both cases, the victims would be justified to consider these acts malicious. If the term 'malicious' still seems too harsh, just consider it to mean the combination of unauthorized and intentional.

Cite this column as follows: Donald Firesmith: "Specifying Reusable Security Requirements", in *Journal of Object Technology*, vol. 3, no. 1, January-February 2004, pp. 61-75.
http://www.jot.fm/issues/issue_2004_01/column6

have. Security requirements are engineered to specify the system's security policies and both policies and requirements should address these security risks. Security mechanisms (e.g., user IDs, passwords, encryption, firewalls, antivirus software, intrusion detection systems, etc.) are then architected to fulfill the security requirements. Some of these concepts influence the engineering of security requirements (e.g., policies, risks, threats, and assets), whereas others (e.g., security mechanisms, security vulnerabilities, and attacks) are influenced by the security requirements.

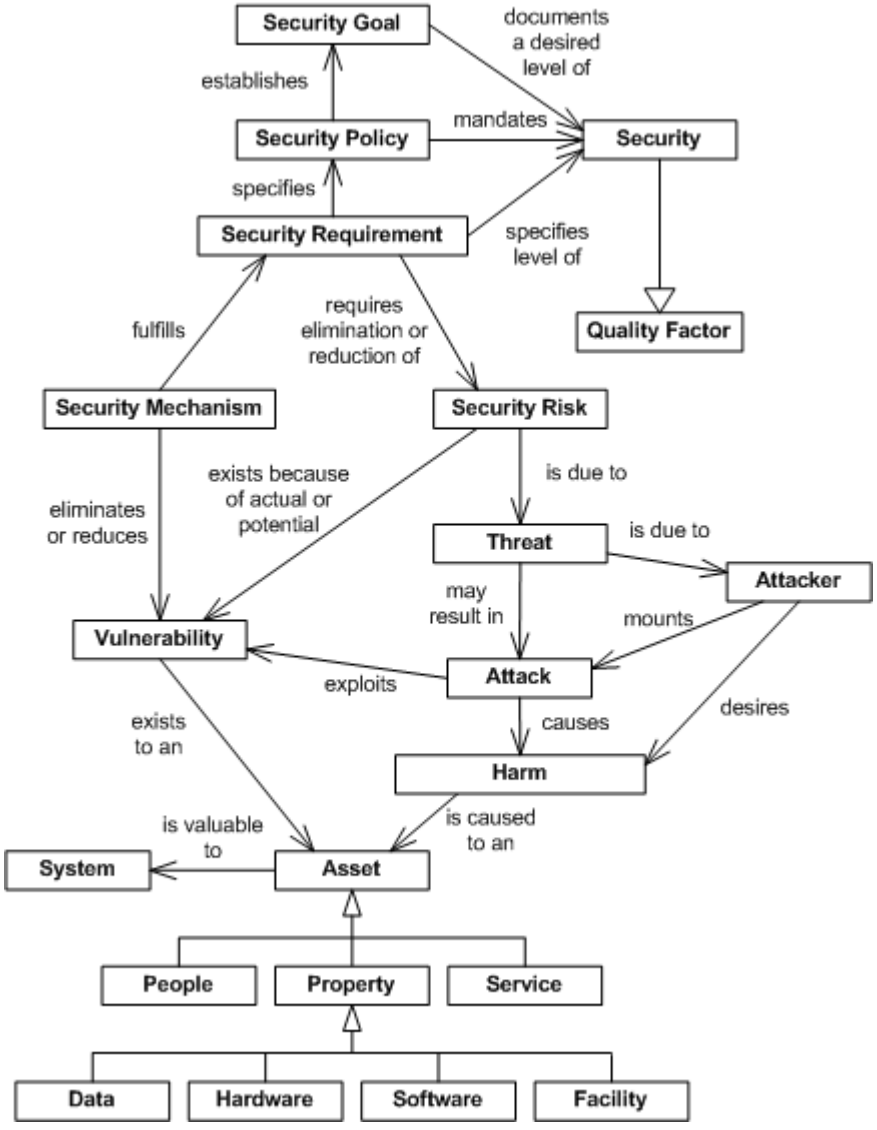
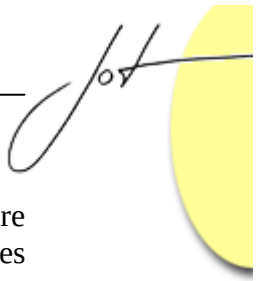



Fig. 1: Concepts that Influence and are Influenced by Security Requirements

The following list defines these security-oriented terms that will be used during the remainder of this column:



- **Asset** is anything of value that should be protected from harm. An asset can require protection because it is the potential target of attack. Assets can be people, properties (e.g., data, hardware, software, and facilities), and services.
- **Attack** (a.k.a., security breach) is an attacker's unauthorized attempt to cause harm to an asset (i.e., violate the security of the system, bypass security mechanisms). An attack may be either successful or unsuccessful.[2]
- **Attacker** is an agent (e.g., humans, programs, processes, devices, or other systems) that causes an attack due to the desire to cause harm to an asset.
- **Harm** is a negative impact associated with an asset due to an attack.
- **Threat** is a general condition, situation, or state (typically corresponding to the motivation of potential attackers) that may result in one or more related attacks.[3]
- **Security** is the degree to which *malicious* harm to a valuable asset is prevented, reduced, and properly responded to. Security is thus the quality factor that signifies the degree to which valuable assets are protected from significant threats posed by malicious attackers.
- **Security Goal** is a quality goal that documents a target level of security or one of its subfactors [Lamsweerde 2000].
- **Security Policy** is a quality policy that mandates a system-specific[4] quality criterion for security or one of its subfactors.
- **Security Mechanism** (a.k.a., countermeasure) is an architecture mechanism (i.e., strategic decision) that helps fulfill one or more security requirements and/or reduces one or more security vulnerabilities. Security mechanisms can be implemented as some combination of hardware or software components, manual procedures, training, etc.
- **Security Requirement** is a quality requirement that specifies a required amount of security (actually a quality subfactor of security) in terms of a system-specific criterion and a minimum level of an associated quality measure that is necessary to meet one or more security policies.
- **Security Risk** is the potential risk of harm to an asset due to the sum (over all relevant threats) of the negative impact of the harm to the asset (i.e., its criticality) multiplied by the likelihood of the harm occurring[5].

[2] Due to their malicious nature, most attacks are cybercrimes, which are crimes (e.g., theft of money or services, fraud, espionage, extortion, vandalism, terrorism, child pornography, etc.) carried out using computer resources. However, some unauthorized misuses of software-intensive systems are merely unethical or malfeasant rather than criminal.

[3] Thus, the threat of theft may result in an actual theft (attack), and threats correspond to attacks that are classified by attacker motivation (e.g., theft) as opposed to technique (e.g., spoofing). In some books and articles, the similar terms 'attack' and 'threat' are confounded by being used as synonyms. [Tulloch 2003]

[4] Remember that this can also involve the system's environment, the infrastructure in which it exists, and any assumptions about the system.

[5] Using the basic theory of conditional probability, the likelihood that harm results from an attack can be calculated/estimated as the product of the following terms: (1) the likelihood that the threat of attack exists, (2) the likelihood that other necessary conditions (e.g., vulnerabilities) also exist, and (3) the likelihood that the threat will lead to a successful attack if it and the other necessary conditions exist. The term likelihood

- **Security Vulnerability** is any weakness[6] in the system that increases the likelihood of a successful attack (i.e., cause harm).

## Quality Subfactors of the Security Quality Factor

Like any other type of quality requirement, security requirements should be based on an underlying *quality model* [IEEE 1992] [ISO 2000a] [Firesmith 2003a]. As previously stated, security signifies the degree to which valuable assets are protected from significant threats posed by malicious attackers. As a *quality factor* (i.e., attribute, characteristic, or aspect), security can be decomposed into a hierarchical taxonomy of underlying *quality subfactors* [Firesmith 2003a] [ISO 2000b] as illustrated in the following figure:

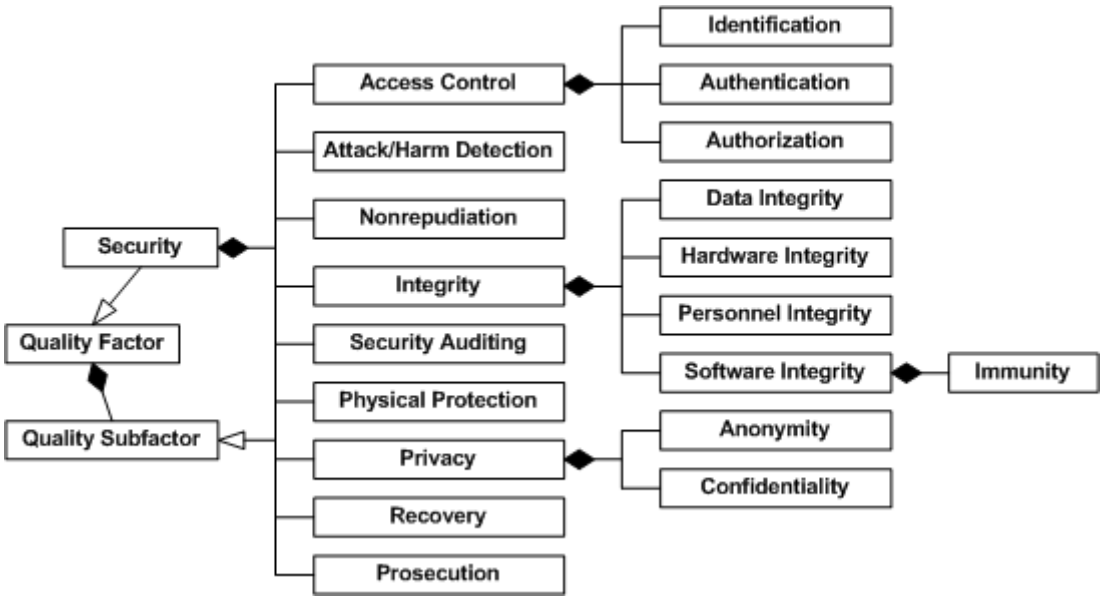



Fig. 2: Taxonomy of Security Quality Factors and Subfactors

These quality factors can be defined as follows:

- **Access Control** is the degree to which the system limits access to its resources only to its authorized externals (e.g., human users, programs, processes, devices, or other systems). The following are quality subfactors of the access control quality subfactor:
  - **Identification** is the degree to which the system identifies (i.e., recognizes) its externals before interacting with them.

is used rather than probability because the probability is typically not known exactly but rather grossly estimated (guesstimated).

[6] This is not restricted to just vulnerabilities due to programming problems. It also includes vulnerabilities in the systems architecture and design, how it is installed and configured, how its users are trained, etc. Remember that the vulnerabilities of a system may involve its hardware components, software components, human role components (a.k.a., wetware or personnel), and document components (i.e., paperware).

  - **Authentication** is the degree to which the system verifies the identities of its externals before interacting with them.
  - **Authorization** is the degree to which access and usage privileges of authenticated externals are properly granted and enforced.
- **Attack/Harm Detection** is the degree to which attempted or successful attacks (or their resulting harm) are detected, recorded, and notified.
- **Integrity** is the degree to which components are protected from intentional and unauthorized corruption:
  - **Data Integrity** is the degree to which data components (including communications) are protected from intentional corruption (e.g., via unauthorized creation, modification, deletion, or replay).
  - **Hardware Integrity** is the degree to which hardware components are protected from intentional corruption (e.g., via unauthorized addition, modification, or theft).
  - **Personnel Integrity** is the degree to which human components are protected from intentional corruption (e.g., via bribery or extortion).
  - **Software Integrity** is the degree to which software components are protected from intentional corruption (e.g., via unauthorized addition, modification, deletion, or theft).
    - **Immunity** is the degree to which the system protects its software components from infection by unauthorized malicious programs[7] (i.e., malware such as computer viruses, worms, Trojan horses, time bombs, malicious scripts, and spyware).
- **Nonrepudiation** is the degree to which a party to an interaction (e.g., message, transaction, transmission of data) is prevented from successfully repudiating (i.e., denying) any aspect of the interaction[8].
- **Privacy** is the degree to which unauthorized parties are prevented from obtaining sensitive information.
  - **Anonymity** is the degree to which the identity of users is prevented from unauthorized storage or disclosure.
  - **Confidentiality** is the degree to which sensitive information is not disclosed to unauthorized parties (e.g., individuals, programs, processes, devices, or other systems).

[7] This includes complete programs, partial programs, processes, tasks, and firmware.

[8] Nonrepudiation covers such information as the identities of the sender and recipient of the transaction, the time and date of the interaction, and any data that flowed with the interaction. Nonrepudiation thus assumes data integrity so that a party cannot argue that the data was corrupted.

- **Security Auditing** is the degree to which security personnel are enabled to audit the status and use of security mechanisms by analyzing security-related events.
- **Physical Protection** is the degree to which the system protects itself and its components from physical attack[9].

## 2 SECURITY REQUIREMENTS

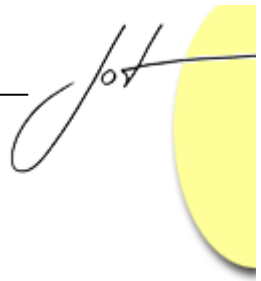
Functionality and its associated functional requirements tend to vary greatly between applications, especially across different application domains. Look at any two requirements specifications, and the main difference between them will almost always be in the content and size of their sections specifying functional requirements. For example, the functional requirements for an embedded avionics application and an ecommerce website may have almost nothing in common. However, the same cannot be said about their security requirements, which tend to exhibit far less variability. After all, both avionics applications and ecommerce applications need to specify levels of identification, authentication, authorization, integrity, privacy, etc.

Every application at the highest level of abstraction will tend to have the same basic kinds of valuable and potentially vulnerable *assets*. Similarly, these assets tend to be subject to the same basic kinds of security *threats* from *attacks* by the same basic kinds of *attackers* who can be profiled with motivations and their typical levels of expertise and tools. Whereas the specific type of attack may vary greatly depending on the architecture under attack, the similarity of threats and attackers tends to lead to considerable uniformity when it comes to the architectural *security mechanisms* that are used to protect these assets from the threats posed by these attackers.

And security requirements tend to be even more standardized than their associated security mechanisms. For any given set of requirements, an architect can and should typically identify and evaluate multiple different architectures and architectural mechanisms before selecting what he or she thinks will be the optimum way of fulfilling the requirements. Thus, there are often many ways for an architecture or security team to address a specific kind of security requirement, and a layered defense will use several of them. For example, to address the identification and authentication (i.e., verification of identification) requirements, one has several choices of architectural mechanisms beyond user IDs and passwords. Specifically, architects and security engineers can base identification and authentication mechanisms on:

- **Who You Say You Are:**
  - Name, user identifier, or national identifier (e.g., social security number).

[9] Physical attack may mean something as violent as the use of a bomb or the kidnapping or blackmailing of personnel. It can also mean something as subtle as the prevention of the theft of a laptop by means of a cable and lock.



- **What You Know:**
  - Your password or personal identification number (PIN).
  - Relatively private personal information such as the last four digits of your social security number, your mother's maiden name, the name of your pet, etc.
- **What You Have:**
  - Digital possessions such as a digital certificate or token.
  - Physical possessions such as an employee ID card, a hardware key, or a smart card enabled with a public key infrastructure (PKI).
- **Who You Are:**
  - Physiological traits (e.g., finger print, palm print, vein pattern, face recognition, iris recognition, and retina scan).
  - Behavioral characteristics (e.g., voice pattern, signature style, and keystroke dynamics).

Yet requirements teams should not constrain the architecture team and security team by specifying unnecessary security architecture mechanisms. Instead, the requirements team should specify what is needed (e.g., a specific required level of identification and authentication) rather than the architectural security mechanisms (e.g., user IDs and passwords) by which they must be achieved. This will result in significantly more uniformity in security requirements than in security mechanisms and associated architectures.

Whereas functional requirements can range over the entire gamut of human imagination, security requirements specify a required amount of a security subfactor and are thus quite limited in scope. One form of reuse of security requirements is then found in the reuse of the security subfactors as a basis for organizing and identifying different kinds of security requirements. For each security subfactor in the quality model, there can be multiple *quality criteria* (i.e., descriptions) that describe the existence of that subfactor and *quality measures* (e.g., the percent of users identified) that can be used to measure the degree of existence of that subfactor [Firesmith 2003b]. A security requirement can thus be defined as a specification of a minimum amount of a security subfactor stated in terms of a security criterion and its associated quality measure. Thus, one form of reuse when specifying measurable and therefore testable security requirements is the reuse of parameterized criteria (e.g., involving the protection of valuable assets from attacks by attackers) and their associated quality measures (i.e., ways to quantify the minimum acceptable level of the criterion).

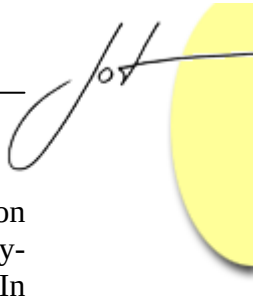
## 3 REUSABLE SECURITY REQUIREMENTS TEMPLATES

Based on the preceding discussion, the following facts combine to strongly argue that highly reusable requirements templates can be produced for reuse across almost all applications and application domains:

- **Requirements**. Security requirements are at a higher level of abstraction than security architectural mechanisms.
- **Security Subfactors**. Only a small number of security quality subfactors exist.
- **Quality Measures**. For each security subfactor, only a small number of associated measures exist.
- **Quality Criteria**. Although there are potentially a large number of application-specific security criteria, these can be parameterized by:
  - **Asset**. The valuable and vulnerable asset to be protected by the security requirement. Although different applications do not have the same assets, they do tend to have the same kinds of assets. The different assets of different applications are subject to different levels of risk.
  - **Threat**. The threat the asset should be protected against. Although different applications are not subject to the same threats, but do tend to be subject to the same kinds of threats (i.e., there are only so many kinds of cybercrimes). And the threats may result in different negative impacts if an associated attack is successful.
  - **Attacker Type**. Different applications tend to be targets of different kinds of attackers with different profiles (e.g., motivation, experience, and resources) who launch different kinds of attacks at different levels of sophistication. Nevertheless, the same kinds of attackers with the same profiles tend to occur over and over.
  - **Situation**. The situation is what tends to be very application specific and is typically the state of the application during the attack, the communications that are occurring, the services that are being requested, the data that are being accessed, and the transactions that are in progress.

Thus, applications tend to have similar classes of security requirements that vary depending on the security subfactor being specified and the associated quality criterion and quality measure chosen to specify the minimum acceptable amount of that security subfactor. Whereas the criticality and specific parameters vary from application to application, a great amount of reuse is available if one has a repository of reusable security requirement templates that formalize the commonality.

This high potential reusability of security requirements is very beneficial because most requirements engineers have had no training in identifying, analyzing, specifying, and managing security requirements and most requirements teams do no include subject matter experts in security. Thus, most current requirements specifications 1) are totally



silent regarding security, 2) merely specify vague security goals such as “The application shall be secure” or “Confidential data shall be kept private.”, or 3) specify commonly-used security mechanisms (e.g., encryption and firewalls) as architectural constraints. In the first case, security too often falls through the cracks and may (or may not) be properly addressed during architecting (or even later when it is much more expensive and difficult to be added to an existing architecture that was not built to support it). In the second case, proper security may still fall through the cracks and often leaves the customer with the unjustified feeling that they have adequately required security even though stating the goal of “The application shall be secure” is hardly a testable requirement. And the third case may unnecessarily tie the architecture team’s hands by specifying an inappropriate security mechanism (e.g., passwords rather than biometrics). Whereas these problems may (or may not) be mitigated by a proper security policy developed by a security team, there is no guarantee that an appropriate security policy will be ready in time to influence the architecture team and the resulting application architecture. And even if the security policy is developed early enough, there is no guarantee that its explicit security policies (and implicit security requirements) will be consistent with the many other functional and quality requirements in the requirements specification. Ultimately, the best approach is to include explicitly specified security requirements with the other quality requirements so that they can all be analyzed, prioritized, and traced, and so that trade-offs can be made to ensure that all requirements are consistent and feasible. Finally, reusable templates for security requirements will help requirements and security teams realize that actual security requirements are both valuable and feasible.

## 4 EXAMPLE TEMPLATE AND ITS USAGE

As an example of a reusable template for specifying security requirements, consider the following template for specifying integrity requirements:

- “The [application / component / data center / business unit] shall protect the [identifier | type] data it transmits from corruption (e.g., unauthorized addition, modification, deletion, or replay) due to [unsophisticated / somewhat sophisticated / sophisticated] attack during execution of [a set of interactions / use cases] as indicated in [specified table].”
  - [Table of interactions / use cases versus minimum acceptable measurement level].

Collaborating with the security team, the requirements team could reuse the preceding template to generate the following integrity requirements:

- “The Global Personal Marketplace (GPM) system shall protect the buyer-related data (see use cases) it transmits from corruption (e.g., unauthorized addition, modification, deletion, or replay) due to unsophisticated attack during execution of the Buyer use cases as indicated in the following table”

| Global Personal Marketplace (GPM) Buyer Use Cases | Minimum Transmissions Protected from Corruption |
|---|---|
| Buyer Buys Item at Direct Sale | 99.99% |
| Buyer Modifies Bid on Item | 99.99% |
| Buyer Modifies Sealed Offer | 99.99% |
| Buyer Places Bid on Item | 99.99% |
| Buyer Places Sealed Offer at Decreasing Price Sale | 99.99% |
| Buyer Reads Buyer Guidelines | 99% |
| Buyer Registers Feedback about Seller | 99.99% |
| … | … |
| GPM Notifies Buyer of Acceptance of Sealed Offer | 99.9% |
| GPM Notifies Buyer of Being Outbid | 99.9% |
| GPM Notifies Buyer of Canceled Sale | 99.9% |
| GPM Notifies Buyer of Relevant Sale | 99.9% |
| GPM Notifies Winning Buyer of Auction Results | 99.9% |

## 5 PROCESS FOR IDENTIFYING AND ANALYZING SECURITY REQUIREMENTS

The preceding sections were primarily concerned with the specification of security requirements using reusable parameterized templates. But how should one determine what actual values to use for these parameters? Typical approaches of requirements analysis (e.g., functional decomposition, use case modeling) were designed for the analysis of functional requirements and are insufficient when analyzing most security requirements. And although misuse cases can help the analyst analyze attacks [Alexander 2001] [Sindre 2001] and security use cases [Firesmith 2003c] can help the analyst understand the system's desired response to these attacks, they still do not adequately support determining the specific values of all of the template's parameters.

So the question remains: how do you fill in the parameters in the reusable templates? It is my contention that any requirements analysis method for security requirements should be asset-based and risk-driven [Alberts 1999]. Different applications have

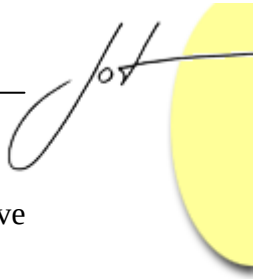
different assets to be protected and failure to protect these assets can result in negative impacts ranging from minor inconvenience to the potential of major lost of life and even war. And different assets will be subject to different kinds of threats due to different kinds of attacks by different kinds of attackers. These threats and negative impacts result in different security risks that need to be addressed.

Given the preceding discussion, we can produce an asset-based risk-driven procedure for the identification and analysis of security requirements. At the highest level, the requirements and security teams could collaborate to perform the following general steps in a highly iterative, incremental, parallel, and time-boxed manner:

1. **Identify Valuable Assets**. Identify the different kinds of valuable assets (e.g., data, communications, services, hardware components, and personnel) that may be subject to security risks. Identification can be based on:
   - The functional, data, and interface requirements
   - Interviews with stakeholders
   - Lists of assets generated during disaster recovery planning
2. **Identify Likely Attacker Types**. Identify the *types* of attackers who most threaten the vulnerable assets. Identification can be based on:
   - Reusable tables of common attacker types
   - Existing reusable attacker profiles (e.g., in terms of attacker motivation, experience level, and resources)
   - The identified valuable assets to be protected
3. **Identify Threats to these Assets**. Identify the general kinds of threats (e.g., theft, vandalism, fraud, unauthorized disclosure, destruction, extortion, espionage, trespass, etc.) to which these assets may be subject. Identification can be based on:
   - The identified valuable assets to be protected
   - The identified types of attackers from which to protect the assets
   - Reusable tables of common threats
   - Essential (i.e., requirements level) misuse cases
4. **Determine Negative Impacts**. For each vulnerable asset, determine the negative impacts which could result if the threats against the asset were to occur.
5. **Estimate and Prioritize Security Risks**. Estimate the security risks to the valuable assets based on the relevant threats and their potential negative impacts. Prioritize these security risks so that the most important ones (e.g., in terms of negative impact and likelihood of occurrence) are handled first given the limited resources of the requirements and security teams.

6. **Select Security Subfactor(s)**. For each important security risk, select the relevant security subfactors from the quality model for which security requirements of that type are needed to limit the risk to an acceptable level.
7. **Select Relevant Template(s)**. For each relevant security subfactor and security risk, select the relevant reusable template(s) for specifying requirements for the relevant security subfactor in terms of the criteria relevant to the security risk. If no relevant template exists, develop one.
8. **Identify Relevant Functional Requirements**. To help identify the relevant security criteria, identify relevant functional requirements based on:
   - The prioritized security risks
   - The selected templates for the selected security subfactors
9. **Determine Security Criterion**. Using the relevant template, determine the appropriate security criterion and enter its parameters into the template. Determination can be based on:
   - The identified valuable assets
   - The identified types of attackers
   - The identified threats
   - The security risks
   - The functional requirements
   - Essential (i.e., requirements level) misuse cases
10. **Determine Quality Measure**. Select the appropriate measure for measuring the existence of the chosen security criterion from the quality model and enter the quality measure into the template.
11. **Determine Required Level**. Based on the security risk to the asset, determine a minimum acceptable level of the measure for that criterion to limit the associated risk to an acceptable level and enter the required level of the measure into the template. A cost-benefit analysis may be used to determine the appropriate level of the quality measure.
12. **Specify Requirement**. Instantiate the template based on the actual parameters from the three previous steps to produce an actual security requirement.

The above process is only one of several that could be used to create security requirements based on the reuse of parameterized templates for security use cases. More important than the specific steps of the preceding process or the order in which these steps are performed is that the requirements and security teams use a documented process with the following useful properties:



- The process should be based on the vulnerable assets to be protected and the negative impacts of the harm that could result if they are not protected.
- The process should achieve high reuse via quality models and parameterized templates.
- The process should enable the requirements and security engineers to determine appropriate values for the parameters in the templates.
- The process should address all significant issues including assets, attackers, threats, negative impacts, and security risks.
- The process should ensure that no important types of security fall through the cracks.

## 6 CONCLUSION

As stated above, one solution to the problem of how to analyze and specify security requirements is for the security team to create and make available a set of parameterized reusable templates that can be used by the requirements team to engineer security requirements that meet the same quality criteria (e.g., correctness, lack of ambiguity, testability) as other requirements. By using a rigorous asset-based risk-driven security requirements analysis method, the quality and appropriateness of the security requirements should rise and the resulting architecture will be more likely to have been designed from the beginning to properly support the security requirements in addition to the functional requirements and the other quality requirements such as availability, capacity, extensibility, internationalization, interoperability, maintainability, performance, portability, reliability, robustness, usability, etc.). The publication of such requirements templates has the potential for greatly improving the quality of security requirements in actual requirements specifications.

## REFERENCES


[Alexander 2001] Alexander, I. “Misuse Case Help to Elicit Nonfunctional Requirements”, *IEE CCEJ*, 2001. Available at http://wwweasyweb.easynet.co.uk/~iany/consultancy/papers.htm

[Alberts 1999] Alberts, C.J. et al., *Operationally Critical Threat, Asset, and Vulnerability Evaluation$^{SM}$ (OCTAVE$^{SM}$) Framework*, Technical Report CMU/SEI-99-TR-017 (1999). Available at http://www.sei.cmu.edu/publications/documents/99.reports/99tr017/99tr017abstract.html

[Firesmith 2003a] *OPEN Process Framework Website*. Available at http://www.donald-firesmith.com/

[Firesmith 2003b] Firesmith, D.G. "Engineering Security Requirements," in *Journal of Object Technology*, vol. 2, no. 1, January-February 2003, pp. 53-68. http://www.jot.fm/issues/issue_2003_01/column6

[Firesmith 2003c] Firesmith, D.G. "Security Use Cases," in *Journal of Object Technology* vol. 2, no. 3, May-June 2003, pp. 53-64. http://www.jot.fm/issues/issue_2003_05/column6

[Firesmith 2003d] Firesmith, D.G. "Analyzing and Specifying Reusable Security Requirements," *Eleventh International IEEE Conference on Requirements Engineering (RE'2003) Requirements for High-Availability Systems (RHAS'03) Workshop*, Monterey, California, September 2003.

[IEEE 1992] IEEE Std 1061-1992, *IEEE Standard for a Software Quality Metrics Methodology*, 1992.

[ISO 2000a] ISO/IEC 9126-1, *Software Engineering – Product Quality – Part 1: Quality Model*, 2000.

[ISO 2000b] ISO/IEC 9126-2, Software *Engineering – Product Quality – Part 2: External Metrics*, 2000.

[Lamsweerde 2000] Axel van Lamsweerde and Emmanuel Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Transactions on Software Engineering*, Vol. 26, No. 10, October 2000, pp. 978-1005

[Sindre 2001] Sindre, G. and Opdahl, A. "Templates for Misuse Case Description," *Seventh International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'2001)*, (4-5 June 2001). http://www.ifi.uib.no/conf/refsq2001/papers/p25.pdf

[Stoneburner 1999] Stoneburner, Gary, *CSPP- Guidance for COTS Security Protection Profiles*, NISTIR 6462, National Institutes of Standards and Technology (NIST), U.S. Department of Commerce, (December 1999), B-4-7. Available at http://csrc.nist.gov/publications/nistir/index.html.

[Tulloch 2003] Mitch Tulloch, *Microsoft Encyclopedia of Security*, Microsoft, Redmond, Washington, 2003.

## ACKNOWLEDGMENTS

This column is an update of a paper [Firesmith 2003d] given at the RE'03 RHAS'03 workshop combined with a subset of the technical note "Common Concepts Underlying Safety, Security, and Survivability Engineering (CMU/SEI-2003-TN-033), December 2003" that I am currently writing for the SEI. The idea of using standardized templates came from [Stoneburner 1999].

## About the author

**Donald Firesmith** is a senior member of the technical staff at the Software Engineering Institute. He has worked exclusively with object technology since 1984 and has written 5 books on the subject. He is currently writing a book on requirements engineering. Most recently, he has developed a 1000+ page informational website on the OPEN Process Framework at http://www.donald-firesmith.com. He can be reached at dgf@sei.cmu.edu.