Analyzing and Specifying Reusable Security Requirements

Donald G. Firesmith
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
+1 412 268-6874
dgf@sei.cmu.edu

ABSTRACT
A system cannot have high assurance if it has poor security, and thus, requirements for high assurance systems will logically include security requirements as well as availability, reliability, and robustness requirements. Unlike typical functional requirements, security requirements can potentially be highly reusable, especially if specified as instances of reusable templates. This paper discusses the value of reusable parameterized templates for specifying security requirements, provides an example of such a template and its associated usage, and outlines an asset-based analysis approach for determining the appropriate actual parameters to use when reusing parameterized templates to specify security requirements.

Keywords
Security requirements, requirements analysis, requirements specification, reusable requirements

1 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. Thus, 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.

At the highest level of abstraction, every application tends to have the same basic kinds of *vulnerable assets* (e.g., data, communications, services, hardware components, and personnel). Similarly, these vulnerable assets tend to be subject to the same basic kinds of security threats (e.g., theft, vandalism, unauthorized disclosure, destruction, fraud, extortion, espionage, trespass, etc.) from attacks by the same basic kinds of *attackers* (e.g., hackers, crackers, disgruntled employees, international cyber-terrorists, industrial spies, governmental spies, foreign military, etc.) who can be profiled with motivations and their typical levels of expertise and tools. Whereas the specific type of *attack* (e.g., password sniffing, spoofing, viruses) 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 (e.g., user IDs, passwords, encryption, firewalls, antivirus software, intrusion detection systems, etc.) 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. 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).
- **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, hand print, 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 and security teams by specifying unnecessary security architecture mechanisms. They should rather specify what is needed (e.g., a certain required level of identification and authentication) rather than the architectural security mechanisms by which they should be achieved. This results in significantly more uniformity in security requirements than in security mechanisms and associated architectures.

Like any other type of quality requirement, security requirements should be based on an underlying *quality model* [1] [2] [3]. Security is a *quality factor* (i.e., attribute, characteristic, or aspect), and it can be decomposed into underlying *quality subfactors* [3] [4] such as:

- **Identification**, which is the degree to which a thing identifies its externals (e.g., human users and external applications) before interacting with them.
- **Authentication**, which is the degree to which something verifies (i.e., confirms) the asserted identity of its externals (e.g., human users and external applications) before interacting with them.
- **Authorization**, which is the degree to which access and usage privileges of authenticated externals exist and are enforced.
- **Immunity**, which is the degree to which a thing protects itself from infection by unauthorized malicious programs (e.g., computer viruses, worms, Trojan horses, and malicious scripts).
- **Integrity**, which is the degree to which communications or [data, hardware, or software] components are protected from intentional corruption (e.g., via unauthorized creation, modification, deletion, or replay).
- **Intrusion Detection**, which is the degree to which attempted or successful access or modification by intruders (i.e., unauthorized individuals or programs) is detected, recorded, and notified.
- **Nonrepudiation**, which is the degree to which a party to an interaction (e.g., message, transaction) is prevented from successfully denying having participated in all or part of the interaction.
- **Privacy (a.k.a., confidentiality)**, which is the degree to which sensitive data and communications are kept private from unauthorized individuals and programs.
- **Security Auditing**, which is the degree to which security personnel are enabled to audit the status and use of security mechanisms by analyzing security-related events.
- **Survivability**, which is the degree to which a thing continues to fulfill its mission by providing essential services in a timely manner in spite of the presence of attacks.
- **Physical Protection**, which is the degree to which a thing protects itself from physical assault.

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 *criteria* (i.e., descriptions) that describe the existence of that subfactor and *measures* (e.g., the percent of users identified) that can be used to measure the degree of existence of that subfactor [5]. 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 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 measures (i.e., ways to quantify the minimum acceptable level of the criterion).

2 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.
- **Measures**. For each security subfactor, only a small number of associated measures exist.
- **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 vulnerability.
  - **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. And the threats may

result in different negative outcomes if an associated attack is successful.

- **Attacker**. 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 is 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 criterion and 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 that "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 specifying that "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 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.

3 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 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 data 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	99.9%

Acceptance of Sealed Offer	
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%

4 REQUIREMENTS ANALYSIS FOR 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 of little use when analyzing most security requirements. And although misuse cases can help the analyst analyze attacks [6] [7] and security use cases [8] can help the analyst understand the system's desired response to these attacks, they still do not adequately support determining the 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 the author's contention that any requirements analysis method for security requirements should be asset-based [9]. Different applications have different assets to be protected and failure to protect these assets can result in negative outcomes ranging from minor inconvenience to the potential of major lost of life and even war. And different assets have different vulnerabilities and are subject do different kinds of threats due to different kinds of attacks by different kinds of attackers. Thus the requirements analysis for security requirements should begin with an asset analysis.

At the highest level, the requirements and security teams could collaborate to perform the following general procedure to analyze security requirements in an 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). Identification can be based on:
   - The functional, data, and interface requirements,
   - Interviews with stakeholders,
   - Lists of assets generated during disaster recovery planning, etc.
2. **Identify Threats**. 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:
   - Reusable tables of common threats and
   - Essential (i.e., requirements level) misuse cases.
3. **Identify Likely Attackers**. Identify the types of attackers who most threaten the vulnerable assets. Identification can be based on:
   - Existing reusable attacker profiles (e.g., in terms of attacker motivation, experience level, and resources).
   - The identified threats to the valuable assets.
4. **Estimate Vulnerability**. Based on their associated threats and attacker profiles, estimate how vulnerable the assets would be to these threats if security requirements are not specified to protect them (i.e., estimate the likelihood of a successful attack). Only gross estimates of relative vulnerabilities may be possible because of limited empirical data.
5. **Determine Negative Outcomes**. For each vulnerable asset, determine the negative outcomes which could result if the threats against the asset were to occur.
6. **Prioritize Vulnerabilities**. Prioritize the vulnerabilities so that the most important vulnerabilities (e.g., in terms of outcome and likelihood) are handled first given the limited resources of the requirements and security teams.
7. **Identify Relevant Situation**. Identify each functional requirement (e.g., use case or use case path) involving the asset and determine if a security requirement is needed.
8. **Consider Security Subfactor**. For each relevant situation, consider each security subfactor from the quality model and determine if a security requirement of that type is needed to limit the associated vulnerability to an acceptable level.
9. **Identify Relevant Template(s)**. For each relevant security subfactor and situation, find the relevant reusable template(s) for specifying requirements for the relevant security subfactor in terms of the criteria relevant to the vulnerability in that situation. If no relevant template exists, the requirements team can analyze the situation using security use cases in order to help produce such a template.
10. **Determine Security Criterion**. For each relevant template, determine the appropriate security criterion and enter its parameters into the template using information about the situation.
11. **Determine Measure**. Select the appropriate measure for measuring the existence of the chosen security criterion from the quality model and enter the measure into the template.
12. **Determine Required Level**. Based on the vulnerability of the asset, select a minimum acceptable level of the measure for that criterion to limit the

associated vulnerability 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 measure.

13. **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:

1. It is based on the vulnerable assets to be protected and the seriousness of the negative outcomes that could result if they are not protected.
2. It achieves high reuse via quality models and parameterized templates.
3. It enables the requirements and security engineers to determine appropriate values for the parameters in the templates.
4. It addresses all significant issues including assets, attackers, threats, outcomes, and vulnerabilities.
5. It ensures that no important types of security

5 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 in a manner similar to other types of requirements. By using a rigorous asset-based security requirements analysis method, the quality and appropriateness of the security requirements will 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.

The members of the RHAS'03 workshop can discuss ways to create, evaluate, and reuse parameterized templates for security requirements. They can also discuss ways to improve the requirements analysis approach for security requirements suggested in this paper.

ACKNOWLEDGEMENTS

This work was triggered as a result of the author discovering Gary Stoneburner's work on security protection profiles [10], which provides a parameterized set of reusable security protection profiles for evaluating the security of COTS software. I would also like to acknowledge discussions with and reviews by Nancy Mead and Carol Woody of the SEI.

REFERENCES

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

2. ISO/IEC 9126-1, Software Engineering – Product Quality – Part 1: Quality Model, 2000.

3. OPEN Process Framework Web Site. Available at <http://www.donald-firesmith.com/>

4. ISO/IEC 9126-2, Software Engineering – Product Quality – Part 2: External Metrics, 2000.

5. Firesmith, D.G. Engineering security requirements. *Journal of Object Technology 2,1* (January-February 2003), 53-68. Available at <http://www.jot.fm/issues/issue_2003_01/column6>

6. Alexander, I. Misuse case help to elicit nonfunctional requirements, IEE CCEJ, 2001. Available at <http://wwweasyweb.easynet.co.uk/~iany/consultancy/papers.htm>

7 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). Available at <http://www.ifi.uib.no/conf/refsq2001/papers/p25.pdf>

8. Firesmith, D.G. Security use cases. *Journal of Object Technology 2,3* (May-June 2003), 53-64. Available at <http://www.jot.fm/issues/issue_2003_05/column6>

9. 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>

10. 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>.