9-2010

Security Requirements Reusability and the SQUARE Methodology

Travis Christian
Carnegie Mellon University

Nancy R. Mead
Carnegie Mellon University, nrm@sei.cmu.edu

Software Engineering Institute

Security Requirements Reusability and the SQUARE Methodology

Travis Christian

Faculty Advisor
Nancy Mead

September 2010

TECHNICAL NOTE
CMU/SEI-2010-TN-027

CERT® Program

http://www.sei.cmu.edu

Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about SEI publications, please visit the library on the SEI website (www.sei.cmu.edu/library).

Table of Contents

List of Tables

Acknowledgments

I would like to thank my instructor, Dr. Nancy R. Mead, a senior member of the technical staff at the Software Engineering Institute and principal investigator for the SQUARE methodology. Her expertise and guidance made this study possible.

Executive Summary

Security is an important and complex quality attribute in many software-intensive systems. Unfortunately security is often neglected in the requirements stage of the development life cycle. Security is introduced later, in design and implementation, which results in inadequate analysis, cost overruns, and vulnerabilities costing billions of dollars annually. Even when security requirements are specified, they are likely at an incorrect level of abstraction, either too general to be useful or too focused on design implications. To be most effective, security should be an integrated part of systems development from the beginning, addressed with the same discipline as other system requirements.

The Security Quality Requirements Engineering (SQUARE) methodology was created by the CERT® Program at Carnegie Mellon University's Software Engineering Institute to address this problem. SQUARE is a nine-step security requirements elicitation approach. Case studies have shown that while SQUARE is effective, the process can take two to three months of effort for large projects, an investment some organizations cannot afford. SQUARE-Lite, a leaner variant, was developed for integration into an existing requirements engineering process. This paper introduces another variant, R-SQUARE, based on SQUARE-Lite, which explores reusable artifacts in various steps to reduce the effort needed to complete SQUARE.

In recent years research has gone several directions with the cataloguing and reuse of requirements and other security-related artifacts. Definitions of a unified security taxonomy for security concepts and terms have also been proposed.

Security requirements, and the goals that produce them, are particularly reusable because security needs and defenses are fairly common across different domains and independent of many functional attributes and other quality requirements. Security requirements and goals can be made very reusable for many different types of projects when they are written at the right level of abstraction, in generic but commonly understood terms, and with good requirements guidelines in mind.

This report introduces a generic security model that attempts to organize various aspects of security into a hierarchy of concepts. Goals, requirements, and other artifacts in SQUARE and beyond can be mapped to corresponding layers of this model, supporting traceability in security analysis and reusability of security statements. The goal of this report is to introduce R-SQUARE, which integrates reusability into the SQUARE methodology in a way that supports collaboration and common understanding.

Abstract

Security is often neglected during requirements elicitation, which leads to tacked-on designs, vulnerabilities, and increased costs. When security requirements are defined, they are often either too vague to be of much use or overly specific in constraining designers to use particular mechanisms. The CERT® Program, part of Carnegie Mellon University's Software Engineering Institute, has developed the Security Quality Requirements Engineering (SQUARE) methodology to correct this shortcoming by integrating security analysis into the requirements engineering process.

SQUARE can be improved upon by considering the inclusion of generalized, reusable security requirements to produce better-quality specifications at a lower cost. Because many software-intensive systems face similar security threats and address those threats in fairly standardized ways, there is potential for reuse of security goals and requirements if they are properly specified. Full integration of reuse into SQUARE requires a common understanding of security concepts and a body of well-written and generalized requirements. This study explores common security criteria as a hierarchy of concepts and relates those criteria to examples of reusable security goals and requirements for inclusion in a new variant of SQUARE focusing on reusability, R-SQUARE.

1 Introduction

Security requirements engineering shows potential for reusing security requirements and goals, which reduces the overall cost of requirements engineering and yields higher-quality requirements specifications. The CERT® Program of Carnegie Mellon University's Software Engineering Institute (SEI) has developed the Security Quality Requirements Engineering (SQUARE) methodology, which can be adapted to support reuse of goals, requirements, and other security artifacts. Such reuse requires a shared understanding of security terminology and concepts at different levels of abstraction and their relation to one another.

This report explores the potential for reuse of security requirements and goals in security requirements engineering and how reuse can be integrated into the SQUARE methodology. A hierarchical security model is described that maps security concepts at different levels of abstraction to the work products created in SQUARE. The report then proposes a new variant of SQUARE, R-SQUARE, that supports the reuse of definitions, goals, risks, and requirements.

Section 2 introduces the state of security requirements and design in industry practice and the problems that led to the creation of SQUARE. Section 3 briefly describes the SQUARE methodology. Section 4 presents an argument for reuse in security requirements engineering, while Section 5 explores related work in requirements engineering and reusable security concepts. Section 6 defines a security model for determining goals, risks, and requirements. Section 7 explains the characteristics of good requirements and goals and how they can be written to be more reusable. Section 8 defines a new variant of SQUARE for reuse, R-SQUARE, built on the concepts presented thus far. Section 9 suggests areas for future work, and Section 10 presents conclusions from the completed research.

® CERT is a registered mark owned by Carnegie Mellon University.

2 Security Requirements in Current Practice

Security requirements engineering is where the creation of a secure system begins. As a quality attribute with major financial, social, and even legal ramifications, security is an integral part of system functionality. As such it must be addressed early in the development process. The cost of catching and correcting requirements defects in deployed systems can be much greater than the cost of doing so during requirements engineering. Defects in requirements, which can account for up to 40–50 percent of project effort, can lead to budget and schedule overruns, quality problems, and canceled projects. This is especially true for security, which has far-reaching implications and great potential for harm if it fails [Mead 2009].

Unfortunately security is often neglected in the requirements stage and only introduced as an afterthought in later stages of development [Schumacher 2006]. Security requirements, if specified at all, face one of two pitfalls. On the one hand, many security requirements are too vague to provide any real guidance to system designers. A generic phrase like "data shall be kept secure" does not communicate anything meaningful [Fabian 2010]. It is left up to architects and programmers, who are disconnected from the business context, to interpret what kind of protection is needed and to what extent. On the other hand, security requirements may overly constrain designers by demanding specific features and mechanisms without defining the underlying need. Finally, security is often ignored altogether until the system is being designed or even implemented. In these cases security features are often tacked onto an existing design. All of these practices lead to unnecessary rework, poor design, and vulnerabilities. Reports have shown that security vulnerabilities cost up to $59.5 billion annually, while early focus on security analysis can provide up to 21 percent return on investment [Mead 2005].

Security requirements engineering suffers from a lack of proper analysis, haphazard specifications, and inadequate management. The industry needs a better way to handle security requirements. To be most effective, they need to be defined at a proper level of abstraction, aligned to the organization's goals, and considerate of the actual risks present. Requirements, which deal with business needs, should address *what* the system should do while leaving the designers to determine *how* to do it. Just like any other quality attribute, security affects system design and functionality. Security needs must be addressed from the beginning of the development life cycle so that they can be integrated into the system design.

3 The SQUARE Methodology

The CERT Program developed SQUARE to address the problem of inadequate security requirements. SQUARE is a requirements engineering methodology "for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications" [Mead 2005]. The process consists of nine steps performed in order by a team of requirements engineers, including at least one expert in risk assessment methods, and project stakeholders:

1. Agree on definitions.
2. Identify assets and goals.
3. Develop supporting artifacts.
4. Perform risk assessment.
5. Select elicitation technique.
6. Elicit security requirements.
7. Categorize requirements.
8. Prioritize requirements.
9. Inspect requirements.

SQUARE is performed at the requirements elicitation stage of the development life cycle to develop security-related system requirements. Engineers guide the stakeholders through the process of identifying and prioritizing security-related goals, threats, and requirements specific to the project. Each step has inputs and outputs, with assigned tasks for one or both. By guiding stakeholders and requirements engineers through the specification of security requirements, SQUARE ensures that security is addressed early in the project life cycle in the same way as functional attributes and other quality attributes.

Completing the full nine-step SQUARE process may take two to three months for large projects. Unfortunately some organizations are unwilling or unable to commit so much time and so many resources to security requirements alone [Gayash 2008]. There is a need for a leaner, more efficient process to achieve similar results. SQUARE-Lite was developed to address this need. SQUARE-Lite is a five-step adaptation that can be completed more quickly and with fewer resources by focusing on only the most essential steps. However, SQUARE-Lite assumes integration into an existing requirements engineering process. SQUARE-Lite consists of the following steps [Mead 2009]:

1. Agree on definitions.
2. Identify assets and security goals.
3. Perform risk assessment.
4. Elicit security requirements.
5. Prioritize requirements.

Reuse of security requirements and other artifacts presents another opportunity to reduce the cost of performing SQUARE. The following sections explore how reusable artifacts can be incorporated into the process with the support of a conceptual model for analyzing security criteria.

4 Advantages of Reuse

Reuse of security requirements provides several benefits to the requirements engineering process.

- Opportunity: Security requirements, more so than other requirements, have potential for reuse in other projects. Many systems face similar security threats and deal with them in the same standardized ways, at least at the requirements level. This is an opportunity to define common security measures and establish reusable artifacts for future projects [Firesmith 2003a].
- Reduced cost: Each time a requirement is reused, it offsets another requirement that does not have to be written. Reuse reduces the effort needed to produce requirements specifications for later projects. Writing requirements that can be reused is a time investment in future productivity.
- Improved quality: A requirement that has been written specifically for reuse will have been given thorough attention and inspected for quality. Reusing published requirements thus results in fewer defects due to poorly written requirements [Firesmith 2003a].
- Consistency: Reusing requirements forces stakeholders to think at the same level of abstraction, in the same terms, and independently of system design in different contexts. Using the same requirement for multiple projects grants a certain level of consistency across a product line or an entire organization [Mellado 2008].
- Less technical knowledge required: Specifying requirements at the correct level of abstraction (by focusing on protection rather than design) lessens the need for security expertise at the requirements stage. A business context and basic understanding of security concepts are sufficient for choosing requirements [Firesmith 2003a].

A set of well-defined security requirements may be reused whole or in part while specifying similar systems. By investigating the goals and concepts behind security requirements and mapping them to common mechanisms and principles, we can see that the primary challenge of eliciting good security requirements is often not specifying innovative solutions but rather deciding what types and degrees of protection are needed.

Reusing predefined criteria does have its disadvantages. There is an up-front investment in effort and resources associated with creating a reusable security repository. Writing goals and requirements for reuse may take some extra thought, and cataloging them for future use will require additional process management. Reusability is unlikely to provide any immediate advantage when it is first adopted.

The real benefit of reusing security requirements comes later when the team can reduce requirements costs by drawing on previous work. Not only will the team become more adept at performing the process, but the actual work to be done will decrease over time. Like the SQUARE methodology itself, adopting reusability is an investment of a little more time now toward reduced costs and higher quality in the future.

5 Related Work

Recent research has gone in several directions to deal with reusability in the context of security analysis and design.

Requirements based on laws and regulations have been the subject of a number of studies. A team in Norway performed such research in reusable security requirements for health care applications, mapping European legal regulations to technical requirements [Jensen 2009]. In Spain another team studied the reuse of legal requirements regarding personal data protection [Toval 2002]. The Privacy Requirements Elicitation Technique, which was incorporated into SQUARE, was supported by a computer-aided software engineering tool that suggested relevant privacy requirements based on a questionnaire about applicable regulations and the use of the system [Miyazaki 2008]. Requirements based on legislation and other standards have great potential for reuse because they essentially reword regulatory language in technical terms.

A study of requirements-based access control and policy specification led to the creation of a support tool that allows analysts to create and reuse rules with traceability to requirements [He 2009]. The analysis itself is based on a three-tiered model of policies, models, and mechanisms, each at a different level of abstraction. This structure influenced the design of the hierarchical security model presented in this report.

Donald Firesmith at the SEI has contributed several ideas to this area of research. He proposes a methodology for reuse-based security analysis, as well as reusable parameterized templates for security requirements elicitation [Firesmith 2003a]. Templates present a solution to some of the problems observed in security requirements. Firesmith also defined a detailed quality model for safety, security, and survivability engineering [Firesmith 2003c]. The model describes relationships between concepts that contribute to systemic qualities.

There have been various other attempts to define a unified model of security concepts [Fabian 2010, Firesmith 2005]. Security lacks a clear taxonomy of attributes, requirements, and standard controls. Inconsistency in language and a focus on controls rather than requirements have hindered a standardized approach to dealing with security needs. There is not yet a consensus about a standardized approach, but the work has helped generate discussion about relationships between abstract security concepts and related design decisions [Blanco 2008].

Another common research topic is requirement repositories for storing reusable requirements [Fabian 2010, Firesmith 2003a]. Architectures and tools have been proposed for cataloguing and tracking reusable requirements so that they can be searched and retrieved easily for later projects [He 2009, Du 2009]. Repositories present an opportunity for collaborative standardization of security requirements that can be used across organizations and domains [Jensen 2009].

6 Defining a Model of Security Concepts

Reuse of security requirements requires a common understanding of the related security concepts [Blanco 2008]. Goals and requirements are given at different levels of abstraction, though the behaviors specified in requirements support the concepts reflected in goals. This observation lends itself to a hierarchical model of security concepts. The model partitions security into a set of high-level concepts called quality subfactors, which represent the characteristics that a secure system exhibits. Quality subfactors are further broken down into security measures, which define general behaviors that support quality subfactors at a level above design patterns and mechanisms. Quality subfactors are reflected in security goals, while requirements are mapped to security measures. The model also includes a layered view of system security, which is helpful when considering different types of security threats and their effects. Many security measures provide defense at a particular layer.

6.1 Quality Subfactors

To understand what security means and define requirements for attaining it, we must understand the specific attributes that contribute to a state of security. In this section we explore some common models for breaking security down into smaller, more specific concepts.

One of the most widely recognized paradigms for analyzing system security is the Central Intelligence Agency (CIA) triad [Pfleeger 2007]. This model defines a secure system as maintaining three properties of its data and services: confidentiality, integrity, and availability. Another quality, accountability, is sometimes included as well. The Department of Homeland Security takes another approach, naming dependability, trustworthiness, and survivability as the essential characteristics of a secure system [Goertzel 2009].

Such attributes are referred to as security properties, principles, or characteristics. As these attributes provide certain systemic properties that fall under the broad quality attribute of security, we will refer to them as security quality subfactors [Firesmith 2003a]. For the purposes of defining security requirements, we will consider the following security quality subfactors:

- confidentiality. Sensitive information must be protected against unauthorized disclosure. Privacy, which is the protection of personal information, is an important subset of confidentiality. Confidentiality protection can take many forms. Data access may be restricted by location, actor identity, or classification level. In the case of personal information (a privacy concern), users may have access to their own data, but no other actor, not even an administrator, may be permitted to see it. Confidentiality may be enforced by hiding data behind a boundary so that unauthorized actors may not access it, encrypting it so that it can only be interpreted by the intended recipient, or a combination of both.
- integrity. Data must be protected against unauthorized modification. Restricting the ability to modify secure data to authorized actors maintains integrity. Integrity may require that data is completely immune to modification or that modification privileges are restricted to certain actors. Access controls and user permissions provide data integrity, and encryption may be used to check for unknown modification. An attribute closely related to integrity is authen-

ticity of data, which means that data is of genuine origin and has not been fabricated. Authentic data has been confirmed to come from a valid source.

- availability. Data and services must be available when they are requested. If interrupted, a system must recover and continue secure operation as quickly as possible without adverse side effects. Availability includes survivability, also known as resilience, which is a system's ability to withstand attacks or accidents and continue to operate in a secure manner.
- accountability. If and when an attack or accident does occur, accountability ensures that actions that affect secure assets can be traced to the responsible actor or condition. Accountability ensures that an attacker who performs a malicious action cannot deny involvement after the fact.
- conformance. While the CIA triad focuses on protection of data, the services dealing with data must also be secure. Conformance means that the software operates as intended without variation. It reliably performs the necessary tasks, no more and no less. The system does not contain vulnerabilities that can be exploited to cause unwanted behavior. Any deviation from the specified behavior constitutes nonconformance because it either produces unwanted behavior or potentially allows an attacker to exploit a vulnerability.

6.2 Security Goals

Security goals are determined first, before requirements, so that the team will understand what outcome the security requirements are to support. As the requirements engineering proverb says, "If you don't know what you want, it's hard to do it right" [Fabian 2010]. Goals define the targeted conditions that make security requirements necessary. While requirements, as we will see later, are phrased in terms of what a system will do, goals focus on the end result. A goal may call for a reduction in damages done, a certain uptime ratio, or other realistic results that business stakeholders are looking for in the implemented system.

Security goals present an opportunity for reuse because systems share similar motivations for implementing security mechanisms. Secure systems are not created simply for the sake of being secure. Security supports the system's primary purpose so that it can be performed in a secure manner. In this sense, security goals are fairly limited: do whatever the system does securely, according to the criteria and extent necessary. Thus, security goals are largely uniform and have a relatively simple purpose: to protect the organization's assets by providing for one or more of the security quality subfactors. As such, goals can be stated in terms of a targeted degree of conformance to a quality subfactor or the targeted business impact of such conformance.

6.3 Layered Defenses

Comprehensive security requires more than a single line of defense. To be most effective, multiple aspects of security should work together to provide layered protection. This is reflected in the security strategy "defense in depth," in which overlapping controls at various layers combine to provide more robust security. There are several layers at which security controls can be implemented, each with its own purpose [Pfleeger 2007, Schumacher 2006]:

- deterrence or prevention. The first layer of defense is to prevent attacks from taking place. An asset can be protected from malicious attack by making the attack as difficult as possible.

One way is to limit the means by which an attacker can gain access to and compromise the asset.

- detection. If an attack or accident does occur, the system and its users should be informed immediately to take action and mitigate the effects. System activities should be monitored for incidents in progress, and notifications should be issued or a response automatically initiated.
- response. When a potential attack or accident is detected, a secure system should take some action to minimize the incident's impact. This may include automated defense mechanisms, active alarms and notifications, or simply recording the incident so that it can be dealt with later.
- recovery. No system is completely secure. Accidents happen and attacks succeed. When an incident occurs and harm is done, action must be taken to recover. Any damages must be corrected, and the system must be returned to secure operation. Recovery services can be defined in terms of what damage the system can correct and how quickly. Because recovery requires a return to secure operation, it is most closely related to availability and specifically survivability.

6.4 Threats

We use the definition of threat from *Software Security Engineering* [Allen 2008]: "A threat is an actor or agent that is a source of danger, capable of violating the confidentiality, integrity, and availability of information assets and security policy." A risk refers to the likelihood that the actor or agent will be successful. Security threat models provide a generalized paradigm for evaluating the different types of security threats and the risks associated with them.

Microsoft's STRIDE Threat Model [Microsoft 2005] is one such example. STRIDE is an acronym for five common categories of threats: Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. Each category presents a risk to one or more security quality subfactor. For example, repudiation violates accountability, and denial of service violates availability.

By using a threat model to categorize and evaluate risks against security goals, an organization can quickly build up a threat database of likely attacks and their potential impact on secure assets. These threats can be described in short scenarios, misuse cases, or any number of formats. A requirements engineering team can then refer to this database, identify relevant threats, and analyze them in terms of the project's own security goals and assets.

6.5 Security Measures

Security measures are the generic, implementation-independent forms of security controls that dictate what the system should do to provide a secure environment. Security measures are not design decisions; rather they describe security in a behavioral sense. Thus, they are at the correct level of abstraction for specifying requirements. There are many types of security measures, whose variety provides for different quality subfactors at different layers. Some of the most fundamental security measures are described here.

- access control. One of the most important and fundamental security measures, access control is a means by which access to a resource is restricted by some condition. Secure systems must ensure that each instance of data or a service is made available only to those who are authorized to access it. Access control makes use of three subsidiary measures to provide secure access to system resources: identification, authentication, and authorization of actors [Schumacher 2006]:
  - identification. Before interacting with an actor, the system must identify it, whether it is a person or another system or component.
  - authentication. When an actor identifies itself, the system must verify the claim against some source known to and trusted by the system. In many cases identification and authentication are done concurrently, as in the case of username and password pairs.
  - authorization. Once the actor's identity has been verified, the system must determine whether the actor is permitted to access the requested data or service. This check may be done on the basis of user roles, user-specific permissions, or some other criteria depending on the system's access control policy. If the system supports user sessions, identification and authentication may be performed only once, while authorization would be performed for each request.
- physical protection. Secure systems must be protected not only from electronic attack but also physical threats. This may include theft, tampering, or destruction of equipment. Physical protection includes a wide variety of defenses against accidents, disasters, and intruders [Pfleeger 2007].
- security policy. A set of rules or practices that a system must enforce is described by a security policy. Policies specify how a system should handle its assets in a secure manner [Schumacher 2006].
- nonrepudiation. Services that provide accountability monitor events and record relevant information about them. When linked to individuals, such data provides nonrepudiation, the inability of an actor to falsely deny involvement in an incident [Schumacher 2006].
- system recovery. An accident or successful attack may compromise the system or its assets in some way. System recovery minimizes the effects of a security failure by restoring the system to a secure state in the case of an attack or accident [Schumacher 2006].
- attack detection. Attack detection is the active or passive monitoring of behaviors and conditions for evidence of an attack [Pfleeger 2007].
- boundary protection. Services that protect the components of a system that are exposed to the outside world limit the means by which an external threat can penetrate the system [Schumacher 2006].

6.6 Relating Levels

Security quality subfactors are the systemic characteristics that define secure behavior. They form the highest level of abstraction in the security model, partitioning the concept of security into separate attributes.

Security goals are statements that describe the system's conformance to quality subfactors. Goals do not specify what exactly the system is to do. Instead, they state that when a system has fulfilled its security requirements, it will exhibit a given set of characteristics.

The system will be exposed to certain threats that can break its conformance to the goals by penetrating security defenses and causing a deviation from secure behavior. The risk that these threats will occur can be mitigated by security requirements that support the provision of characteristics stated in the security goals. Requirements are made in terms of security measures, which describe the type of defense to be implemented without constraining the design to specific mechanisms. In other words, requirements dictate what the system must do but not how to do it. Requirements mandate the presence of defenses around secure assets. Such defenses are best applied in layers so as to maximize the robustness and coverage of security mechanisms.

Security mechanisms come out of the resulting design decisions that are made in light of the requirements. Mechanisms are concrete patterns and techniques such as password protection, secure sockets layers (SSL), or firewalls.

Thus, each level's specification brings the system closer to a concrete design and provides a context for the next level of concepts. The following example, taken from a SQUARE case study with some alteration to support reusability, illustrates this point. Each level of the security model is represented in one of the statements. As SQUARE progresses from goals and assets to risks to requirements, it moves down the model from abstract quality subfactors to more concrete but still generic security measures.

Step 2: Identify assets and goals

Assets: *User data, hardware, application software*

Business goal: *The tool provides the means to make informed decisions based on available sources.*

Security subgoal: *The confidentiality and integrity of the user data shall be maintained against unsophisticated login attack 99 percent of the time.*

Step 4: Risk assessment

Threat: *An attacker succeeds in an unauthorized server login (low sophistication), resulting in loss of data integrity and confidentiality.*

Step 6: Elicit security requirements

Requirement: *The system is required to have strong authentication measures in place at all system gateways/entrance points.*

From this breakdown, we can see that the stated goals have led to a requirement, which will lead to specific design decisions. The specified requirement mitigates the threat and supports the security goal. The goals, risks, and especially the requirements are each stated at a proper level of abstraction and in a way that supports reuse.

7 Writing Reusable Requirements

Now that we have explored security concepts at different levels of abstraction, we can see how they relate to the artifacts defined in requirements analysis. We turn our attention specifically to writing requirements that support reuse. What makes a security requirement a good candidate for reuse? It must have the qualities of a good requirement in the first place, and it also must be general enough to apply in different situations while carrying the same meaning.

If a requirement is to be placed in a collective body of knowledge and used again in other contexts, some care must be taken that it is well written and exhibits the qualities that a requirement should carry. The total impact of a reused requirement over its lifetime will be much greater than that of a single-use requirement.

A reusable requirement also must not be overly specific to the system in question, or it will lose its value outside of that context and cannot be reused without significant rewording. Requirements that must be significantly modified to be meaningful in another project are not reusable and may as well be discarded.

We will first explore the characteristics of a good requirement and then see what can be done to make them suitable for reuse. There are various criteria for what a good requirement should look like [Firesmith 2003b]. For example, in the context of security engineering, Pfleeger and Pfleeger propose six characteristics for requirements: correctness, consistency, completeness, realism, need, verifiability, and traceability [Pfleeger 2007]. They also state that security requirements should cover all aspects of security and be stated in a way that specifies function without constraining design.

7.1 Quality Criteria for Requirements

The following criteria have been found to characterize good requirements and specifications [Firesmith 2003b, Pfleeger 2007]. Requirements that meet all of these criteria will be good candidates for reuse because they will be sufficiently abstract to apply to multiple projects while clearly communicating security needs.

- atomic. The requirement addresses exactly one point and stands on its own as a single, complete thought in one statement. Generally speaking, an atomic requirement should not contain more than one clause.
- complete. Collectively, the requirements provide a full representation of system functionality. For security requirements, this means full coverage of the identified goals and risks. All relevant aspects of security, including each affected quality subfactor and layered defense, are fully specified at a proper level of abstraction.
- consistent. The requirement does not conflict with or confuse the meaning of any other requirement. The same function is not described in different terms or from different perspectives. The entire specification provides a single, uniform model of functionality.
- feasible. The requirement can be reasonably implemented within the project's constraints. Requirements do not represent wishful thinking, recommendations, or unrealistic goals.

- generic. Requirements should communicate a system's functions, not the means by which those functions are achieved. A good requirement does not constrain the design by imposing a particular mechanism. To put it another way, requirements state "what," not "how."
- necessary. The requirement communicates exactly what is needed and no more. It is an essential statement without which the project's specification would be incomplete. There are no redundant requirements, and overlap is kept to a minimum. Each requirement provides a unique piece of information.
- traceable. The requirement can be related to decisions and products throughout the development life cycle so that its implementation can be verified at any point and any changes to the requirement can be easily assessed.
- unambiguous. The requirement is worded in a way that is clear and difficult to misunderstand. The requirement avoids undefined jargon and vague terms and allows only one objective interpretation.
- verifiable. The requirement can be verified through testing or other forms of analysis. A system characteristic that cannot be proven is a goal, not a requirement.

Any requirements specification should meet the above criteria. The first step in writing reusable requirements is to write *good* requirements. Now we turn our attention to those characteristics that specifically make a requirement suitable for reuse.

7.2 Need for Common Terminology

Reusable requirements will be used outside of their original context. They will be applied to different projects while retaining the same meaning. This calls for common terminology in defining security requirements. SQUARE begins by agreeing on definitions. For requirements reuse to be feasible, teams need to agree on definitions not only for a single project but across projects in the organization. Agreeing to use widely accepted definitions from a reputable public source is even better. IEEE, the Software Engineering Body of Knowledge (SWEBOK),[1] and the U.S. Department of Defense are all potential sources of defined terminology.

7.3 Generic but Useful Requirements

Requirements tend to be stated in terms specific to the project for which they are written. But reusable requirements need to retain their meaning outside of their original context. A requirement written for reuse should not refer to any specific system component or feature. Rather it should state a systemic trait that is to be implemented. If it is not possible to write a meaningful requirement in generic terms, the requirement may not be written at a sufficient level of abstraction, or it may be a special case not suitable for reuse.

For example, this might be a fine requirement:

The system must authenticate users arriving at the control panel.

But what does it mean in a generic context? This requirement is only meaningful when there is a control panel at which a user can arrive. For reusability, the requirement would be better stated as

[1] http://www.computer.org/portal/web/swebok

The system is required to have strong authentication measures in place at all system gateways/entrance points.

This requirement is independent of any specific system features and retains its meaning in almost any context.

7.4 Right Level of Abstraction

Requirements are most useful when they specify what a system must do without placing any constraints on how that behavior is achieved. A requirement that enforces a technique or mechanism puts unnecessary limitations on designers, who must create a design that conforms to the stated requirements.

It can be tempting to begin design of security features while specifying requirements. If accountability is an important quality subfactor, a requirement might state

The system shall require users to log in with a username and password combination.

This is a standard way to force authentication. But why must the login mechanism use a username and password? What if these users are in a hurry and just want to swipe a card or scan their fingerprint? Those may be valid alternatives that achieve the same end result: user authentication. The requirement is better stated as

The system shall implement access control via a secure login screen.

This requirement states the intended security measure and the point at which it is implemented without specifying a certain design.

Another example shows that requirements can also be too abstract:

The system's availability shall be maintained in the event of a denial-of-service attack 99 percent of the time.

This makes a fine security goal, but it is too general to be of much use as a requirement. Specifically, it is not verifiable—a successful denial-of-service attack would have to be defined and the associated figure of 99 percent availability measured in order to verify this requirement. A better requirement would state what the system must do to achieve the desired level of availability:

The system shall recover from attacks, failures, and accidents in less than one minute.

This requirement speaks to the same quality subfactor but is more specific to system behavior. It places a clear, testable measurement to be achieved by one of the layers of security (recovery) without constraining any particular design decision. This requirement can be reused with practically any system.

8 Integration into SQUARE

The motivation for integrating reusability into SQUARE is twofold. First, it is easier to select and adapt existing requirements from a repository than to write new ones. Reusing requirements will reduce the total effort needed to perform SQUARE, especially as a practicing organization develops or acquires a large body of knowledge from which to draw requirements, goals, and risks. Second, by focusing their attention on reusing and refining the same well-defined requirements multiple times, organizations can produce higher-quality requirements and create better specifications overall. Rather than always writing and reviewing new requirements, engineers can choose from predefined requirements whose quality has already been verified.

With these goals in mind, a new variant of SQUARE, R-SQUARE, is defined using SQUARE-Lite as a base model and incorporating reuse in several places (see Table 1). This new version includes six steps, five of which provide some opportunity for reuse of artifacts and concepts.

Table 1: R-SQUARE Steps

	Step	Input	Techniques	Opportunities for Reuse	Participants	Output
1	Agree on definitions	Security glossary and/or definitions from external standards	Work session	Definitions	Stakeholders, requirements engineers	Agreed-on definitions
2	Identify assets and security goals	Definitions, predefined and candidate goals, security quality model, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Security- and business-oriented goals	Stakeholders, requirements engineers	Assets and goals
3	Perform risk assessment	Misuse cases, scenarios, threat models, security goals		Threat models, attack trees, documented risks	Requirements engineer, risk expert, stakeholders	Risk assessment results
4	Choose security requirements	Definitions, goals, risks, pre-defined requirements, templates	Work session, focus groups, checklists, lists of reusable requirements	Requirements	Stakeholders facilitated by requirements engineers	Initial cut at security requirements
5	Prioritize requirements	Requirements, risks	Prioritization methods such as Analytical Hierarchy Process (AHP), triage, Win-Win		Stakeholders facilitated by requirements engineers	Prioritized requirements
6	Review requirements	Prioritized requirements, review techniques	Inspection method such as Fagan, peer reviews	New requirements	Review team	Initial selected requirements

8.1 Step 1: Agree on Definitions

SQUARE begins with all participants agreeing on definitions to be used in the subsequent steps. Requirements engineers, customers, security specialists, and other stakeholders may use the same terms but mean different things or refer to the same concepts using different words. To communicate clearly and effectively, everyone involved in requirements engineering must agree to a set of definitions to be used from this point onward.

Agreeing on definitions is especially important when considering reusable requirements because the same wording will be used in different contexts. Any definitions used in reusable requirements should hold beyond the scope of the current project. An organization that implements reusable requirements will need to create and maintain a glossary of relevant terms and definitions so that the meanings of requirements do not become ambiguous over time as they are reused. After a glossary is established, it will be important to minimize changes to existing definitions so that reused goals and requirements retain their intended meaning. Some or all stakeholders may need to adapt their terminology for the purpose of conforming to the wording of existing defined terms.

This step obviously requires some additional investment if the organization does not have a resource glossary, but the extra time spent on its creation will pay off in future projects. Instead of debating and writing new definitions each time the process is followed, the team can simply choose the applicable predefined terms from the organization's glossary. Of course, a team that has performed SQUARE in the past is likely to refer back to previously agreed-on definitions whenever possible, but the emphasis here is to intentionally define terms independently of the current project.

Alternatively, an organization can borrow many or all of its security definitions from established sources such as the SWEBOK or IEEE. Whether using internal or external sources, certain projects may involve specific terminology that is not a part of the glossary, in which case the team will have to establish those definitions on its own. The important thing is that terms are used consistently across projects and teams so that reused statements retain their meaning and avoid ambiguity.

8.2 Step 2: Identify Assets and Goals

Step 2, Identify Assets and Goals, calls for the identification of valuable assets and security goals for the project. This step ensures that security requirements will reflect the organization's policies and priorities for protecting its assets. Security goals should be aligned to the project's essential quality subfactors and any business-oriented objectives for meeting them.

Each stakeholder may have a slightly different focus, which should be reflected in the goals. For example, a stakeholder in human resources may be more interested in preserving the privacy of user information, while a financial executive will be concerned with the integrity of accounting data. Both have valid security implications. Each stakeholder's concerns, along with the overall business context, should be considered when selecting goals [Mead 2005].

An organization may have one or more top-level security goals for all of its projects. It may also have several applicable subgoals depending on the level of assurance desired and the types of

threats present. Organizations that develop product lines of secure software will likely have overarching business and security-related goals that are intended to apply to all affected projects.

After goals are identified, they must be prioritized, either by consensus or executive decision.

8.3 Step 3: Risk Assessment

In Step 3, Risk Assessment, the team analyzes relevant security risks in light of business and security goals. In this step the team will identify risks and determine how they affect each of the goals and associated security quality subfactors.

The risk assessment should employ an expert in risk assessment methods, who may recommend a particular method based on the organization's needs. Goals identified in Step 2 and their associated security quality subfactors provide the input to risk assessment. Depending on the method chosen, the team may make use of existing risk-related artifacts such as misuse cases or historical security data.

Threat models, which are abstract and highly reusable, can be used to identify relevant threats and map them to the project's security quality subfactors. Previously defined risks may also apply, though the team will need to reassess their impact for the new project.

8.4 Step 4: Choose Requirements

Step 4, Choose Requirements, is the core of SQUARE. This is when the team actually decides on the requirements that will go into the system specification. Requirements are chosen by stakeholders based on the previously identified goals and the specific risks that pose a threat to the related quality subfactors. Stakeholders identify suitable requirements from the repository or write new ones as needed. A more rigorous elicitation process will be needed when writing new requirements to ensure their quality. This obviously will require more investment in the first several projects (unless the organization opts to borrow predefined requirements, if a suitable compilation exists), but over time the organization can begin to rely more on the accumulated work of past projects.

Any number of techniques may be suitable for choosing requirements. SQUARE has been performed with several defined candidate processes [Chung 2006]. For selecting predefined requirements, an informal method such as focus groups or stakeholder interviews may be appropriate. Some requirements may have been or need to be sanitized by removing or changing specific wording to make them more applicable to the project's context.

The team should take care to include requirements that address each of the identified quality subfactors pertaining to the security goals. If stakeholders have difficulty choosing appropriate requirements, it may help to work backward by examining applicable security mechanisms and tracing them back to related requirements. This tactic allows stakeholders to think in terms of common security concepts without unduly constraining system designers with overly specific requirements.

8.5 Step 5: Prioritize Requirements

Step 5, Prioritize Requirements, is not affected by reuse. The team prioritizes requirements so that management can evaluate tradeoffs and know which requirements to focus on if resources become scarce. The same security requirements may have different relative priorities in different settings, depending on the applicable threats and the level of assurance required, so it is meaningless to assign an inherent priority outside of some context. Prioritization should be done independently for each project, regardless of whether the requirements are new or reused.

Prioritization requires the involvement of both the requirements engineering team and stakeholders. Stakeholders consider the business consequences of having or not having certain security measures in place, while engineers may focus on the technical risks and costs involved with implementing them. The requirements engineering team should perform cost-benefit analysis, if possible, to aid stakeholders in their decision making.

SQUARE does not prescribe any one way of prioritizing requirements, although the Analytical Hierarchy Process (AHP) has been found to be effective. AHP is a pair-wise comparison method that uses multiple factors for estimating value. In the case of requirements, this is often cost versus benefit. Various other options are explored in SQUARE case studies, including triage, Win-Win, and mathematical models. Prioritization may be done by either numerical ranking or classification (essential, conditional, optional, etc.).

8.6 Step 6: Review Requirements

Step 6, Review Requirements, is the final step to conclude security requirements engineering. At this point a full set of requirements has been defined and prioritized along with associated goals and risks. Review of reused requirements should take less effort than the inspection step mandated by the full version of SQUARE because at least some requirements will have already been inspected when they were created. The focus of this step is rather to demonstrate that the correct set of requirements has been selected. Of course, any new requirements should be given full attention, especially if they are candidates for reuse.

At a minimum, any new requirements should be evaluated in terms of the identified characteristics of good requirements. If requirements have been previously used and published for reuse, it can be reasonably assumed that they are of high quality and do not need to be individually inspected in detail. However, a requirements specification is more than the sum of its parts. Three characteristics in particular—completeness, consistency, and necessity—must be assessed collectively. A review of the entire set of requirements will reveal whether these characteristics have been met. When reviewing requirements, the team should refer back to the results of previous steps to check that everything has been accounted for. The chosen requirements should fully address business and security goals, risks, and threats.

After passing a review, the security requirements are ready to be incorporated into a requirements specification or other permanent artifact. Any new requirements that meet reusability criteria can be submitted to the repository. At this point, the SQUARE process is complete, and the organization will be well on its way to developing more secure systems using reusable security requirements and goals.

9 Recommendations for Future Work

This report has presented a unified, structured model for thinking about software security from quality attributes down to control mechanisms. This model was the basis for defining a new variant of the SQUARE methodology with reuse of artifacts. The security model and its mapping among concepts is a start, but more work is needed to map out the concepts in each layer and the relationships between layers.

Both the conceptual model and the new process draw heavily from previous research and case studies, but neither has yet been applied to a real project. The next step is to field-test R-SQUARE in a case study and compare the results with those of SQUARE and SQUARE-Lite. From there, research into the actual process of creating and managing reusable artifacts can improve on the methodology.

10 Conclusions

The ideas presented in this report demonstrate that reusability is a viable extension to the SQUARE process. A new variant of SQUARE has been defined that specifically calls for consideration of reuse in several of its steps. The incorporation of reuse into SQUARE has the potential to reduce the cost of performing the process repeatedly. A conceptual model that defines related ideas at multiple levels of abstraction aids the reuse of goals, requirements, and threats. Such a model, backed by standardized definitions, is needed to support a common understanding of security concepts in the context of reusable artifacts.

Appendix A: Concepts from the Security Model

Table 2 and Table 3 present a survey of known relationships among security subfactors and measures identified in this report as well as common mechanisms. This information is intended to help relate requirements to goals and design decisions to requirements.

Table 2: Security Quality Subfactors and Associated Measures

Security Quality Subfactor	Associated Security Measures
Confidentiality	Access control Physical protection Security policy
Integrity	Access control Nonrepudiation Physical protection Attack detection
Availability	System recovery Physical protection Attack detection
Accountability	Nonrepudiation Attack detection
Conformance	Access control Physical protection Attack detection

Table 3: Security Measures and Associated Mechanisms

Security Measure	Associated Security Mechanisms
Access control	Biometrics Certificates Multilevel security Passwords and keys Reference monitor Registration Time limits User permissions VPN
Security policy	Administrative privileges Malware detection Multilevel security Reference monitor Secure channels Security session Single access point Time limits User permissions VPN
Nonrepudiation	Administrative privileges Logging and auditing Reference monitor
Physical protection	Access cards Alarms Equipment tagging Locks Offsite storage Secured rooms Security personnel
System recovery	Backup and restoration Configuration management Connection service agreement Disaster recovery Off-site storage Redundancy
Attack detection	Administrative privileges Alarms Incident response Intrusion detection systems Logging and auditing Malware detection Reference monitor
Boundary protection	DMZ Firewalls Proxies Single access point VPN

Appendix B: Examples of Reusable Goals and Requirements

Reusable Goals

Table 4 presents example security goals from SQUARE case studies [Chung 2006] that are potentially reusable.

Table 4: Potentially Reusable Security Goals

Goal	Quality Subfactors
Management shall exercise effective control over the system's configuration and usage.	Conformance
The confidentiality, accuracy, and integrity of the system's data shall be maintained.	Confidentiality Integrity
The system shall be available for use when needed.	Availability

Reusable Requirements

Table 5 presents example requirements from SQUARE case studies [Chung 2006] that are potentially reusable.

Table 5: Potentially Reusable Requirements

Requirement	Security Measures	Related Quality Subfactors
The system is required to have authentication measures in place at all gateways/entrance points.	Access control (authentication) Boundary protection	Integrity Confidentiality Accountability
The system is required to have a role-based access control mechanism that governs which system elements (data, functionality, etc.) users can view, modify, and/or interact with.		
It is required that a continuity of operations plan (COOP) be in place to ensure system availability.	Security policy System recovery	Availability
It is required that designated security personnel be able to audit the status and usage of system resources (including security devices).	Nonrepudiation	Accountability
Designated personnel are required to audit the status of system resources and their usage on a regular basis.	Security policy	Accountability
It is required that the system's network communications be protected from unauthorized information gathering and/or eavesdropping by encryption and other reasonable techniques.		Confidentiality
It is a requirement that both process-centric and logical means be in place to prevent the installation of any software or device without prior authorization.	Physical protection Boundary protection Security policy	Conformance
It is required that physical devices be protected against destruction, damage, theft, tampering, or surreptitious replacement (including but not limited to damage due to vandalism, sabotage, terrorism, or natural disaster).	Physical protection	Availability
The website shall ensure the integrity of content that is provided to the users by using authentication, authorization, and access control.	Access control (authentication, authorization)	Integrity
The website shall enable auditing features that log all content modifications, work-flow state transitions, access failures, and authentication attempts.	Nonrepudiation	Accountability
The website shall ensure that only authenticated users can access its protected content.	Access control (authentication)	Confidentiality Integrity
The website shall protect the authenticated users' privacy by securing the communication channel.	Access control (authentication)	Confidentiality (privacy)

Glossary of Terms

access control
Security measure; access to a resource that is restricted to those who are authorized.

accountability
Security subfactor; the ability to trace actions affecting a secure resource to the responsible actor or condition.

attack detection
Security measure; the active or passive monitoring of behaviors and conditions for evidence of an attack.

authentication
Security measure; verification by the system of a claim of identity or origin against some source known to and trusted by the system. Authentic data is confirmed to have come from a valid source.

authorization
Security measure; a part of access control in which the system determines whether the actor is permitted to access the requested data or service.

availability
Security subfactor; the presence and accessibility of data and services when they are requested. If interrupted, a system recovers and continues secure operation as quickly as possible without adverse side effects.

boundary protection
Security measure; services protecting the components of a system that are exposed to the outside world. Boundary protection limits the means by which an external threat can penetrate the system.

confidentiality
Security subfactor; the protection of sensitive information against unauthorized disclosure. This includes privacy, the protection of personal information.

conformance
Security subfactor; the operation of software as intended and without variation. It reliably performs the necessary tasks, no more and no less. The system does not contain vulnerabilities that can be exploited to cause unwanted behavior.

detection
Layer of defense; the monitoring of system activities for incidents in progress. Upon detecting an incident, the system will issue a notification or automatically initiate a response.

deterrence
Layer of defense; the protection of assets from malicious attack by making the attack as difficult as possible. The means by which an attacker can gain access to and compromise the asset are limited. Also known as prevention.

identification
Security measure; a part of access control in which the system identifies an actor before interacting with it.

integrity
Security subfactor; the protection of data against unauthorized modification and fabrication.

nonrepudiation
Security measure; the monitoring of events and recording of relevant information to disprove an actor's false denial of involvement in an incident.

physical protection
Security measure; protection from physical threats such as theft, tampering, or destruction of equipment, including defenses against accidents and disasters.

privacy
The protection of personal information; an aspect of confidentiality.

recovery
Layered defense; actions taken to correct any damage done and return to secure operation after a harmful incident.

response
Layered defense; actions taken to minimize an incident's impact when a potential attack or accident is detected.

security measure
A generic, implementation-independent form of security control that dictates what the system should do to provide a secure environment. It describes security in a behavioral sense, not as a design decision.

security policy
Security measure; a set of rules or practices that a system must enforce. It specifies how a system should handle its assets in a secure manner.

security quality subfactor
A specific systemic property under the security quality attribute that contributes to a state of security.

survivability
A system's ability to withstand attacks or accidents and continue to operate in a secure manner; an aspect of availability. Also known as resilience.

system recovery
Security measure; services that minimize the effects of a security failure by restoring the system to a secure state during or after an attack or accident.

References

URLs are valid as of the publication date of this document.

[Allen 2008]
Allen, J.; Barnum, S.; Ellison, R.; McGraw, G.; & Mead, N. *Software Security Engineering: A Guide for Project Managers*. Addison-Wesley, 2008.

[Blanco 2008]
Blanco, C.; Lasheras, J.; Valencia-Garcia, R.; Fernandez-Medina, E.; Toval, A.; & Piattini, M. "A Systematic Review and Comparison of Security Ontologies," 813-820. *2008 Third International Conference on Availability, Reliability and Security* (ARES '08). Barcelona, Spain, March 2008. IEEE Computer Society, 2008.

[Chung 2006]
Chung, Lydia; Hung, Frank; Hough, Eric; & Ojoko-Adams, Don. *Security Quality Requirements Engineering (SQUARE): Case Study Phase III* (CMU/SEI-2006-SR-003). Software Engineering Institute, Carnegie Mellon University, May 2006. http://www.sei.cmu.edu/library/abstracts/reports/06sr003.cfm

[Du 2009]
Du, Jing; Yang, Ye; & Wang, Qing. "An Analysis for Understanding Software Security Requirement Methodologies," 141-149. *Third IEEE International Conference on Secure Software Integration and Reliability Improvement*. Shanghai, China, July 2009. IEEE Computer Society, 2009.

[Fabian 2010]
Fabian, Benjamin; Guerses, Seda; & Heisel, Maritt. "A Comparison of Security Requirements Engineering Methods." *Requirements Engineering 15*, 1 (March 2010): 7-40.

[Firesmith 2003a]
Firesmith, Donald G. "Analyzing and Specifying Reusable Security Requirements" (slides), 7-11. *Requirements Engineering 2003 Requirements for High Assurance Systems (RHAS) Workshop Proceedings*. Monterey, CA, September 2003. IEEE Computer Society, 2003.

[Firesmith 2003b]
Firesmith, Donald G. "Specifying Good Requirements." *Journal of Object Technology (JOT) 2*, 4 (July-August 2003): 77-87.

[Firesmith 2003c]
Firesmith, Donald G. *Common Concepts Underlying Safety, Security, and Survivability Engineering* (CMU/SEI-2003-TN-033). Software Engineering Institute, Carnegie Mellon University, December 2003. http://www.sei.cmu.edu/library/abstracts/reports/03tn033.cfm

[Firesmith 2005]
Firesmith, Donald G. *A Taxonomy of Security-Related Requirements*. Software Engineering Institute, Carnegie Mellon University, 2005.
http://www.sei.cmu.edu/library/abstracts/whitepapers/taxonomysep2005.cfm

[Gayash 2008]
Gayash, Ashwin; Viswanathan, Venkatesh; & Padmanabhan, Deepa. *SQUARE-Lite: Case Study on VADSoft Project* (CMU/SEI-2008-SR-017). Software Engineering Institute, Carnegie Mellon University, June 2008. http://www.sei.cmu.edu/library/abstracts/reports/08sr017.cfm

[Goertzel 2009]
Goertzel, Karen Mercedes. *Introduction to Software Security*.
https://buildsecurityin.us-cert.gov/bsi/547-BSI.html (2009).

[He 2009]
He, Q. I. & Antón, A. I. "Requirements-Based Access Control and Policy Specification (ReCAPS)." *Information and Software Technology 51*, 6 (June 2009): 993-1009.

[Jensen 2009]
Jensen, J.; Tondel, I. A.; Jaatun, M. G.; Meland, P. H.; & Andresen, H. "Reusable Security Requirements for Healthcare Applications," 380-385. *International Conference on Availability, Reliability and Security (ARES '09)*. Fukuoka, Japan, March 2009. IEEE Computer Society, 2009.

[Mead 2005]
Mead, Nancy R.; Hough, Eric; & Stehney, Theodore R., II. *Security Requirements Engineering (SQUARE) Methodology* (CMU/SEI-2005-TR-009). Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/library/abstracts/reports/05tr009.cfm

[Mead 2009]
Mead, Nancy R. *SQUARE Overview*. http://www.sei.cmu.edu/library/assets/20090514webinar.pdf (2009).

[Mellado 2008]
Mellado, D.; Fernandez-Medina, E.; & Piattini, M. "Towards Security Requirements Management for Software Product Lines: A Security Domain Requirements Engineering Process." *Computer Standards & Interfaces 30*, 6 (August 2008): 361-371.

[Microsoft 2005]
Microsoft. "The STRIDE Threat Model." *MSDN Library*.
http://msdn.microsoft.com/en-us/library/ee823878(CS.20).aspx (2005).

[Miyazaki 2008]
Miyazaki, Seiya; Mead, Nancy; & Zhan, Justin. "Computer-Aided Privacy Requirements Elicitation Technique," 367-372. *2008 IEEE Asia-Pacific Services Computing Conference*. Yilan, Taiwan, December 2008. IEEE Computer Society, 2008.

[Pfleeger 2007]
Pfleeger, Charles P. & Pfleeger, Shari Lawrence. *Security in Computing*, 4th ed. Prentice Hall, 2007.

[Schumacher 2006]
Schumacher, Markus; Fernandez-Buglioni, Eduardo; Hybertson, Duane; Buschmann, Frank; & Sommerlad, Peter. *Security Patterns: Integrating Security and Systems Engineering.* John Wiley & Sons, 2006.

[Toval 2002]
Toval, Ambrosio; Olmos, Alfonso; & Piattini, Mario. "Legal Requirements Reuse: A Critical Success Factor for Requirements Quality and Personal Data Protection," 95-103. *Proceedings of the IEEE Joint International Conference on Requirements Engineering* (RE'02). Essen, Germany, September 2002. IEEE Computer Society, 2002.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2010	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Security Requirements Reusability and the SQUARE Methodology	FA8721-05-C-0003

6. AUTHOR(S)

Travis Christian, Nancy Mead (faculty advisor)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2010-TN-027

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Security is often neglected during requirements elicitation, which leads to tacked-on designs, vulnerabilities, and increased costs. When security requirements are defined, they are often either too vague to be of much use or overly specific in constraining designers to use particular mechanisms. The CERT® Program, part of Carnegie Mellon University's Software Engineering Institute, has developed the Security Quality Requirements Engineering (SQUARE) methodology to correct this shortcoming by integrating security analysis into the requirements engineering process.

SQUARE can be improved upon by considering the inclusion of generalized, reusable security requirements to produce better-quality specifications at a lower cost. Because many software-intensive systems face similar security threats and address those threats in fairly standardized ways, there is potential for reuse of security goals and requirements if they are properly specified. Full integration of reuse into SQUARE requires a common understanding of security concepts and a body of well-written and generalized requirements. This study explores common security criteria as a hierarchy of concepts and relates those criteria to examples of reusable security goals and requirements for inclusion in a new variant of SQUARE focusing on reusability, R-SQUARE.

14. SUBJECT TERMS	15. NUMBER OF PAGES
security requirements engineering, reuse, software engineering	44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL