

CMU/NCSU Collaborative Proposal: Improving the Usability of Security Requirements by Software Developers through Empirical Studies and Analysis

Travis Breaux

Institute for Software Research
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
breaux@cs.cmu.edu

Laurie Williams

Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
williams@csc.ncsu.edu

Jianwei Niu

Department of Computer Science
University of Texas – San Antonio
San Antonio, Texas, USA
niu@cs.utsa.edu

Hard Problem: Security requirements are difficult to apply in design and must incorporate system architecture, functional requirements, security policies, regulations, and standards.

Summary: Secure software depends upon the ability of software developers to respond to security risks early in the software development process. Despite a wealth of security requirements, often called security controls, there is a shortfall in the adoption and implementation of these requirements. This shortfall is due to the extensive expertise and higher level cognitive skillsets required to comprehend, decompose and reassemble security requirements concepts in the context of an emerging system design. To address this shortfall, we propose to develop two empirical methods: (1) a method to derive security requirements patterns from requirements catalogues using expert knowledge; and (2) a method to empirically evaluate these patterns for their “usability” by novice software developers against a set of common problem descriptions, including the developer’s ability to formulate problems, select and instantiate patterns. The study results will yield a framework for discovering and evaluating security requirements patterns and new scientific knowledge about the limitations of patterns-based approaches when applied by novice software developers.

Project Description

The security of information systems has long been a concern for software developers. Over the past 25 years, extensive security guidance and resources have been developed and made available to software developers. This guidance includes the National Institute of Standards and Technology (NIST) Special Publication 800-53 catalog of security controls, the ISO/IEC 15408:2005 or Common Criteria functional requirements, and industry-specific guidance, such as the Payment Card Industry Data Security Standards, which contains over 100 functional requirements for payment processors (PCI-DSS). Recent U.S. Federal and state regulations, such as Massachusetts and Nevada data security laws and the U.S. Health Insurance Portability and Accountability Act (HIPAA) Security Rule, impose industry-wide functional security requirements on financial and healthcare systems, including encryption, single-user logon and threat monitoring. Unlike guidance documents, these regulations change infrequently and risk adhering to malpractice¹ or security technologies of the day. Thus, a benefit to society is an improved ability by software developers to apply security best practices to avoid the costly burden of well intended, but outdated government regulation.

Despite the wide availability of security best practices, we continue to observe security failures in software systems. Breaux and Baumer report an empirical study of 10 years of regulatory enforcement actions by the U.S. Federal Trade Commission (FTC) and found that companies largely violated their privacy and security policies by failing to implement sufficient security controls [6]. Their analysis surfaced 39 security requirements, including electronic and physical controls, which were described as the factual basis for these legal complaints. In all cases, the FTC recommends companies establish an active security program with periodic monitoring and outside reviews. The cost of applying these practices afterward a security failure can be severe: in the case of ChoicePoint in 2006, the re-engineering costs alone exceeded millions of dollars [25].

Before developers can mitigate the risks of security threats, they must know the requirements for the system's security. The challenge with security requirements is translating *non-functional* security attributes (e.g., trusted identity, encryption, resiliency, etc.) to *functional* requirements that define specific functions or behaviors of the system. Functional requirements statements often specify desired system behavior in "shall" statements [32], for example: "The system *shall* send an email message to the administrator containing the new user name and the time and date of creation when a new account is created." Effective security requires that we validate not only that the system does what it should, securely, but also that the system does not do what it is not intended to do [31]. To address this unintended functionality, methods have been developed to guide developers in eliciting and documenting security requirements. These include framework-methods, such as the SQUARE method, which provide a Capability Maturity Model-like reference model for coordinating various technical activities [28], and new technical activities and artifacts, such as misuse or abuse cases [30], anti-goals [24], and assurance arguments [17].

Our position is that software developers have difficulty recognizing when to apply these requirements to their information systems. The NIST, ISO and PCI control catalogs provide lists of requirements grouped by category, but lack the conceptual scaffolding to enable software developers to identify when a particular requirement applies to their system design. In terms of Bloom's Taxonomy, we explain this difficulty by recognizing that

¹ Massachusetts state law 201 CMR 17.02 requires companies to use *encryption* in which meaning cannot be assigned without a confidential process or key; this includes any simple substitution cipher which is generally accepted as malpractice.

highest levels of learning include Analysis and Synthesis, or breaking wholes into parts and recombining these parts into new wholes [4]. Similarly, Alistair Cockburn identifies the highest levels (2 and 3) of software developer competence as the ability to mix and match elements of a process or to construct their own patterns [5]. These highest levels are typically represented in only a small percentage of professionals [5]. Thus, our research goal is to develop an empirical framework to enable security experts to synthesize security requirements into functional patterns that can be applied by software developers with skills at the middle and lower levels of Bloom's Taxonomy and Cockburn's expertise levels. These middle and lower levels include developers with the skillset to apply patterns to problem descriptions.

Design patterns have long been recognized as artifacts for generalizing best practices in software development [16] and building architecture [2]. Alexander introduced the notion of design pattern languages [1], which were tailored to software engineering by Coplien [12]. A pattern language is a collection of patterns that build on each other to generate a software system [1]. A pattern language is functionally complete, meaning that using one pattern creates an imbalance that is resolved by another pattern, and so on until a whole system can be developed that is balanced and well-designed [12]. In requirements engineering, Jackson introduced problem frames as a means to characterize requirements patterns [20]. Breaux et al. show how grounded analysis can be applied to a set of requirements to identify several patterns that can be used to refine regulatory codes into functional requirements [8]. We propose to extend this prior work in two ways through the development of our framework: (1) developing an empirical method for deriving security requirements patterns from requirements catalogues using expert knowledge; and (2) conducting empirical studies of novice software developers in problem formulation, pattern selection and instantiation. These two studies will extend our knowledge of usable security – not the traditional usability concerns that arise when a user attempts to configure their system to set an appropriate level of security, but the usability of security requirements as developers select requirements to design systems to support a range of security levels.

In this research, we plan to analyze existing security requirements catalogues using the *grounded analysis* research methodology, which includes techniques for coding data and addressing threats to validity with the goal of deriving new theory from a dataset [13]. The new theory produced by this empirical method will consist of our security requirements patterns. To illustrate, consider the security requirements presented in Table 1. These requirements state positive actions for a software designer: *implement data retention* in PCI-DSS 3.1.1, or *sanitize media* in SP 800-53 control MP-6. What is missing from these requirements, and what a pattern would provide to the developer, are queues about what actions the developer must take to implement these requirements. These queues are based on contextual information that is specific to the system under development. The queues target information observables that the designer would be more familiar with, thus improving the usability of security knowledge by grounding the knowledge in the context of specific design.

In Figure 1, we present a hypothetical security requirement pattern that generalizes and parameterizes critical concepts from the requirements in Table 1 into a set of logical queues and corresponding questions for the developer. The queues include: *activation*, which includes system events that cause this pattern to become relevant to the designer; *outcomes*, which describe the actions taken by implementing the pattern; *triggers*, which link state changes in the system to pattern outcomes; and *extension points*, which include questions that must be answered to manage the pattern's outcomes. For example, suppose data was no longer needed by a system, because a payment transaction was completed and the requested product or service was delivered, then the default outcome may be to dispose

data. However, the extension points could interact to change the default outcome. For example, if this data is subject to a law enforcement investigation, state law may require the data to be retained until the investigation is completed. Such laws would extend this pattern, by adding additional conditions and events that affect the retention and disposal timeline. Moreover, the category of risk to the data and the type of preferred disposal method may change, depending on which laws and standards govern the data. Changes to these extensions may be imposed externally: laws change, or standards are updated to address emerging technology and evolving security threats. By retaining these patterns and their instantiations for a specific system, we foresee the ability to monitor externally changing requirements after the system has been delivered so that developers can plan updates, accordingly.

Table 1. Example security requirements from PCI-DSS and SP 800-53 related to data retention and disposal

Payment Card Industry Data Security Standard (PCI-DSS) Version 2	NIST Special Publication 800-53
3.1.1 Implement a data retention and disposal policy that includes: ▪ Limiting data storage amount and retention time to that which is required for legal, regulatory, and business requirements ▪ Processes for secure deletion of data when no longer needed ▪ Specific retention requirements for cardholder data ▪ A quarterly automatic or manual process for identifying and securely deleting stored cardholder data that exceeds defined retention requirements	AU-11: Audit record retention MP-6: Media sanitization SI-12: Information output handling and retention

Figure 1. Example security requirements pattern corresponding to requirements in Table 1.

Pattern Name: Retention and Disposal Pattern

Pattern Activation: Data is received, stored or processed

Pattern Outcomes: Retain data; Dispose data

Pattern Triggers:

- The data is no longer needed
- Physical or digital access to the media will change

Mandatory Extension Points:

- When was the data acquired?
- What laws, regulations or business requirements exist to retain the data?
- How will the data be destroyed?
 - Option A: De-reference (Low risk)
 - Option B: Secure wipe (Medium risk)
 - Option C: Incinerate (High risk)

As the basis of conducting this research, we assume that existing catalogues of requirements contain a significant and broad representation of security requirements and concerns. However, we recognize these catalogues often represent mainstream or fundamental requirements intended for a broad class of systems, and thus emerging technologies, such as mobile or cloud computing, may be unforeseen or underrepresented

by our resulting dataset.

The second study consists of assessing existing empirical methods for evaluating security requirements pattern applications by software developers. This study includes analysis of current research in cognitive theory on the human ability of engineers to express problems and apply patterns. This study will be sensitive to identify sources of error across the steps of problem formulation, pattern selection and instantiation with an aim to modify our pattern presentations to counteract this error. We assume that existing literature on cognitive theory is mature and will inform our investigation, whereas a shortfall in existing cognitive theory would require an extension to this project and collaboration with an expert in cognitive psychology to develop new theories. We plan to conduct these studies using undergraduate and graduate software engineering students with varying levels of expertise and practical experience across all participating institutions. The studies will be seeded with problem descriptions that we will survey empirically through analysis of reported security incidents and interviews with subject-matter experts (SMEs). We plan to leverage our ongoing relationship with CMU's Computer Emergency Response Team (CERT) to gain access to SMEs for this survey. Additionally, we will conduct case studies with industrial partners of North Carolina State University.

Significance

Security as an intrinsic property of a system is difficult to measure. Security methodologies recommend that software developers perform security-enriching processes such as the development of security requirements [28], penetration testing [3], threat modeling [19], automated static analysis [15], testing access control policies [10, 22], risk analysis and misuse cases [30], black box security testing [27], and many other techniques. The concept of *building security in* prescribes that developers consider system security from the outset of the project and design the system to be protected from malicious attack [26]. Each of these techniques plays a role in the prevention and removal of vulnerabilities, but none of these techniques will find all vulnerabilities (i.e., there is no silver bullet) [27].

A recent survey of information security and software development consultants indicates that although 81% of respondents were aware of formal secure development methodologies, but 43% indicated that they were not using any methodology. Additionally, these methodologies rarely offer detailed advice on *how* to contextualize security requirements in the context of their systems. For example, security experts use their extensive knowledge and experience to reason about and identify possible attacks on an application in an exploratory and opportunistic way in a process known as penetration testing [3]. However, the success of current security assurance techniques like penetration testing that occur late in the product's lifecycle vary based on the skill, knowledge, and experience of testers and provide little support to designers earlier in the lifecycle [3]. Additionally, methodologies such as SQUARE [28] or Microsoft's Secure Software Development Lifecycle [18] do not provide prescriptive guidance on the specifics of what is typically included in a comprehensive set of security requirements. Security requirements patterns can provide such guidance.

The immediate impact of our research proposal will be threefold: (1) new security requirements patterns to map security requirements to problem descriptions; (2) knowledge of the efficacy of developers in their application of the patterns; and (3) a framework through which new patterns can be discovered and further refined through experimental evaluation in a laboratory and industrial case study setting. As new technologies give rise to new security threats, security experts will discover new

requirements to avoid, mitigate or reduce these threats. We envision our proposed framework as a reusable methodology to guide these experts in identifying new patterns from these emerging requirements. For example, we envision applying this framework to emerging technologies, such as mobile or cloud computing, or to domain-specific applications, such as electronic voting, electronic medical record systems, or supervisory control and data acquisition (SCADA) systems, as each of these would introduce their own unique requirements and lead to developing new patterns.

Collaboration, Schedule and Timeline

The proposed research is a cross-lablet collaboration between Dr. Travis Breaux at Carnegie Mellon University and Dr. Laurie Williams at North Carolina State University. Dr. Breaux has expertise in the empirical analysis of privacy and security requirements to develop reusable ontologies [6, 7], formal methods for requirements analysis [9] and requirements patterns to aide developers in refining regulatory codes into product requirements [8]. Dr. Williams has extensive expertise in empirical methods for evaluating software developers [33] and in the empirical development of software test patterns [11]. In addition, we plan to collaborate externally with Dr. Jianwei Niu at the University of Texas at San Antonio. Dr. Niu has extensive expertise in scenario modeling [14, 21], trust management [29] and privacy and security policy modeling [23]. The laboratory studies would be replicated across all participating institutions.

The research is planned as a multi-year effort (see Table 1). After the completion of the first year, we expect to have several deliverables, including documented security patterns and the scientific evidence to inform the design of our user studies. This evidence includes knowledge of the distribution of security failures to which we can apply our patterns and results from pilot studies that inform how we present the patterns and problems to software developers. In year two, we plan to design and execute the user studies and summarize our results.

Table 1. Schedule of Work and Deliverables

Timeframe	Work Item
Jan-Jun 2012	- Survey cognitive theory and pattern literature for empirical study design guidance (Williams) - Select and analyze security requirements catalogues (Breaux, Niu)
Jul-Dec 2012	- Summarize pattern findings in conference papers (Breaux, Williams, Niu) - Design pilot studies to evaluate pattern execution alternatives (Williams)
Jan-Jun 2013	- Execute pilot studies to evaluate user study design alternative (Williams) - Survey problem descriptions based on incident reports and interviews with subject matter experts (Breaux, Niu) - Design user studies with students and practitioners to evaluate pattern usability (Williams)
Jul-Dec 2013	Conduct and replicate user studies at NCSU and CMU (Williams, Breaux, Niu)

Spring 2014	• Conduct analysis of user study data (Williams, Breaux, Niu) • Summarize study findings in conference and journal papers (Williams, Breaux, Niu)
----------------	--

During the course of this effort, we plan to host three in-person working sessions, alternating between our host institutions (NCSU and CMU):

- (1) Pattern Workshop: in early Fall 2012, a session to present the patterns and discuss directions for the problem survey
- (2) Pilot Results Workshop: in late Spring 2013, a session to present the pilot study results and discuss the user study designs; and
- (3) Usability Design Teleconference: in late Summer 2013, conduct a teleconference to coordinate usability study replications.
- (4) Usability Results Workshop: in late Spring 2014, a final session to review the analysis results from the user studies.

Additional teleconferences will be held monthly or every two months, as needed.

References

- [1] C. Alexander, *A Pattern Language: Town, Buildings, Construction*, Oxford, UK: Oxford University Press, 1977.
- [2] C. Alexander, *The Timeless Way of Building*, Oxford, UK: Oxford University Press, 1979.
- [3] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security & Privacy*, vol. 3, no. 1, pp. 84-87, 2005.
- [4] Benjamin S. Bloom and David R. Krathwohl (ed). "Taxonomy of educational objectives: The classification of educational goals." *Handbook I, cognitive domain*. Longmans, Green, 1956.
- [5] B. Boehm, R. Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, 2003.
- [6] T.D. Breaux, D.L. Baumer. "Legally 'Reasonable' Security Requirements: A 10-year FTC Retrospective." *Computers & Security*, 30(4), 178-193, 2011.
- [7] T.D. Breaux, A.I. Antón. "Analyzing Regulatory Rules for Privacy and Security Requirements." *IEEE Transactions on Software Engineering, Special Issue on Software Engineering for Secure Systems*, 34(1):5-20, Jan/Feb 2008
- [8] T.D. Breaux, A.I. Antón, K. Boucher, M. Dorfman. "Legal Requirements, Compliance and Practice: An Industry Case Study in Accessibility." *IEEE 16th International Requirements Engineering Conference*, Barcelona, Spain, pp. 43-52, Sep. 2008
- [9] T.D. Breaux, A.I. Antón, J. Doyle. "Semantic Parameterization: A Process for Modeling Domain Descriptions." *ACM Transactions on Software Engineering Methodology*, 18(2): 5, November 2008
- [10] A. D. Brucker, L. Brügger, P. Kearney, and B. Wolff, "An approach to modular and testable security models of realworld health-care applications," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*, Innsbruck, Austria, 2011.
- [11] Smith, B. and Williams, L., *On the Development of A Black Box Security Test Pattern Catalog Based on Empirical Data*, submitted to the International Conference on Software Engineering (ICSE) 2012.
- [12] J. Coplien, *Software Patterns*, New York, NY, USA: SIGS Books & Multimedia, 2000.
- [13] J. Corbin, A. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, Sage Publications, 2007.
- [14] S. Esmaeilsabzali, N.A. Day, J.M. Atlee, J. Niu. "Deconstructing the Semantics of Big-Step Modeling Languages", *Requirements Engineering*, 15(2), pages 235-265, April 2010.

- [15] D. Evans, and D. Larochelle, "Improving Security Using Extensible Lightweight Static Analysis," *IEEE Software*, vol. 19, no. 1, pp. 42-51, 2002.
- [16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Boston, MA: Addison Wesley Longman Publishing Company, 1995.
- [17] V.N.L. Franqueira, T.T. Tun, Y. Yu, R. Wieringa, B. Nuseibeh. "Risk and argument: A risk-based argumentation method for practical security." *IEEE International Requirements Engineering Conference*, 2011, pp. 239-248
- [18] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA: Microsoft Press, 2006.
- [19] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover Security Design Flaws Using the STRIDE Approach," *MSDN Magazine*, <http://msdn.microsoft.com/enus/magazine/cc163519.aspx>, 2006].
- [20] M. Jackson. *Problem Frames*. Addison-Wesley, 2001.
- [21] J. Niu, J.M. Atlee, N.A. Day. "Template Semantics for Model-Based Notations", *IEEE Transactions on Software Engineering*, 29(10), pages 866-882, October 2003
- [22] J. Julliand, P.-A. Masson, and R. Tissot, "Generating security tests in addition to functional tests," in *Automation of Software Testing (AST)*, Leipzig, Germany, 2008, pp. 41-44.
- [23] R. Krishnan, J. Niu, R. Sandhu, W.H. Winsborough. "Group-Centric Secure Information Sharing Models for Isolated Groups", To Appear: *ACM Transactions on Information and System Security*, November 2011.
- [24] A. v. Lamsweerde, "Elaborating Security Requirements by Construction of Intentional Anti-Models," in *International Conference on Software Engineering (ICSE 2004)*, Edinburgh, Scotland, 2004, pp. 148-157
- [25] A. Litan, "Case Study: ChoicePoint Incident Leads to Improved Security, Others Must Follow," *Gartner*, September 19, 2006.
- [26] G. McGraw, *Software Security: Building Security In*: Addison-Wesley Professional, 2006.
- [27] G. McGraw, and B. Potter, "Software Security Testing," *IEEE Security and Privacy*, vol. 2, no. 5, pp. 81-85, 2004.
- [28] N. R. Mead, and T. Stehney, "Security quality requirements engineering (SQUARE) methodology," in *Software Engineering for Secure Systems (SESS)*, St. Louise, Missouri, USA, 2005, pp. 1-7.
- [29] M. Reith, J. Niu, W.H. Winsborough. "Towards Practical Analysis for Trust Management Policy", *4th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 310-321, March 2009.
- [30] G. Sindre, and A. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Engineering*, vol. 10, no. 1, pp. 34-44, 2005.
- [31] H. H. Thompson, and J. A. Whittaker, "Testing for software security," *Dr. Dobb's Journal*, vol. 27, no. 11, pp. 24-34, 2002.
- [32] K. E. Wiegers, *Software Requirements, 2nd Edition*, Redmond, WA: Microsoft Press, 2003.
- [33] L. Williams, C. McDowell, N. Nagappan, J. Fernald, and L. Werner, "Building Pair Programming Knowledge Through a Family of Experiments," in *International Symposium on Empirical Software Engineering (ISESE) 2003*, Rome, Italy, 2003, pp. 143-152.