Generative Secure Design, Defined

Riccardo Scandariato
University of Gothenburg
Gothenburg, Sweden
riccardo.scandariato@cse.gu.se

Jennifer Horkhoff
University of Gothenburg
Gothenburg, Sweden
jennifer.horkhoff@cse.gu.se

Robert Feldt
Chalmers
Gothenburg, Sweden
robert.feldt@chalmers.se

ABSTRACT

In software-intensive industries, companies face the constant challenge of not having enough security experts on staff in order to validate the design of the high-complexity projects they run. Many of these companies are now realizing that increasing automation in their secure development process is the only way forward in order to cope with the ultra-large scale of modern systems. This paper embraces that viewpoint. We chart the roadmap to the development of a generative design tool that iteratively produces several design alternatives, each attempting to solve the security goals by incorporating security mechanisms. The tool explores the possible solutions by starting from well-known security techniques and by creating variations via mutations and crossovers. By incorporating user feedback, the tool generates increasingly better design alternatives.

CCS CONCEPTS

• **Security and privacy → Software security engineering**;

KEYWORDS

Security, Design, Tool

ACM Reference Format:
Riccardo Scandariato, Jennifer Horkhoff, and Robert Feldt. 2018. Generative Secure Design, Defined. In *ICSE-NIER'18: 40th International Conference on Software Engineering: New Ideas and Emerging Results Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183399.3183400

1 AN ANALYSIS OF SECURE DESIGN TODAY

Secure design requires security experts, which are hard to come by. The research community has developed techniques and tools to deal with security in the early phases of the software development, including design. However, most of these techniques are not geared towards software engineers, but rather rely on the presence of a security expert with the necessary experience and knowledge. Empirical research has shown that secure design methods, such as security threat analysis [16] or modeling with security patterns [17], are labor-intensive and difficult for regular engineers to apply. During our collaboration with companies of all sizes, we observed a general shortage of security experts. The resources allocated to security assurance activities in a software development project are very limited. Anecdotal accounts report an average of about 3-5% of the overall budget being set aside for security. As a result, experts have little time for security design validation. Budget is not the only constraint. Even when the budget is extended to recruit security experts, it is difficult to find qualified individuals.

Unrealistic tasks, even for experts. At the same time, trends such as the Internet of Things, micro services, and Big Data have acted as enablers for the development of ultra-large scale systems. Consequently, the complexity of modern systems is growing beyond what a human expert can possibly handle. It is becoming impossible for security experts to get the necessary understanding of how the system is structured overall, how the parts interact with each other and with third parties, what assumptions can be safely made about the operating environment, and what security measures should be realistically incorporated into the system design.

Not ready for continuous delivery. Finally, security in design is often addressed in separate steps: the design is first defined from a functional and business perspective, followed by a security assessment resulting in the subsequent retrofitting of the design with security measures. This way of operating is suitable for products that have slow-paced update cycles. The rise of continuous delivery requires a more agile model where there is no longer a rigid separation between designers and security experts.

2 VISION

This paper argues that iterative, user-driven automation is the key factor for achieving three objectives: 1) dealing with the ultra-large scale of modern systems, 2) countering the shortage of security experts, and 3) bringing security into the design process from the beginning, hence overcoming the retrofitting phenomenon.

Generative design. This paper is inspired by generative design, an approach that stems from the field of product design and is here extended to the field of software security. Generative design adopts an evolutionary approach based on search and optimization strategies such as genetic algorithms. In its current use, designers of material objects input their design goals (e.g., a light-weight sport shoe for joggers), the materials to be used, the manufacturing costs, and so on. The generative design algorithms assist the designer in the form-finding process. Starting from well-known solutions, the algorithms explore new possibilities and create design variations by means of mutations and crossovers, in a manner analogous to biological evolution. The process uses feedback from the designer to iterate through several generations until variations are found that satisfy the design goals effectively. The quick generation of design alternatives is essential to rapid prototyping, which is important for objects, but also for software.

ICSE-NIER'18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5662-6/18/05…$15.00
https://doi.org/10.1145/3183399.3183400

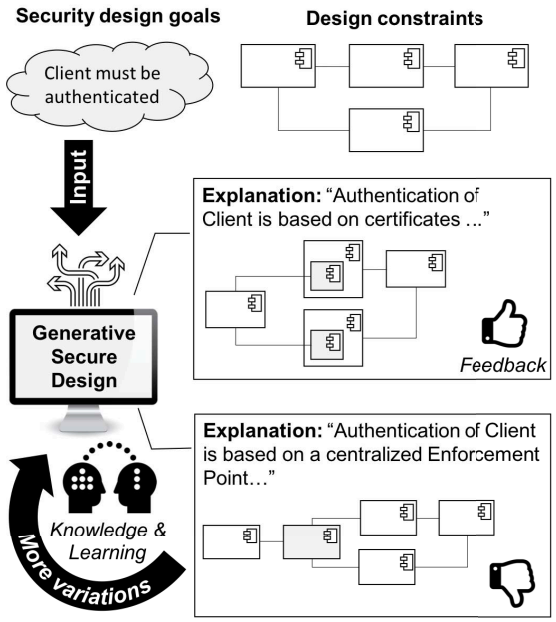

Figure 1: Generative Secure Design in Action.

Generative Secure Design, defined. As we will discuss further in Section 4, techniques based on evolutionary algorithms have already been used to a limited extent to improve parts of software development processes and support engineers, e.g., to automate the generation of test cases. In this paper, we define an innovative tool-assisted techniques inspired by generative design, applied for the first time to secure software design, for which we coin the term Generative Secure Design (GSD). The approach is illustrated in Figure 1. The tool-supported approach takes the security design goals as input, along with existing design constraints, such as an existing draft software architecture. To avoid clutter, the figure shows only one goal, but in a real usage scenario we assume that there are multiple secure design goals, which need to be accommodated jointly. The tool would start from known secure design solutions (e.g., reference architectures for access control such as Shibboleth Single Sign-On, or security patterns [18]) and evolve them to produce several initial variations of the design, each attempting to solve the security goals in a different way. The tool *explains to the designer the differences* between the alternatives. In turn, the designer indicates which alternatives they prefer, providing rationale for their decisions. This allows the tool to *learn the user preferences* in each iteration, enabling the generation of incrementally better design alternatives until the designer is satisfied. Over time, this accumulated knowledge allows the tool to increase its company-specific expertise, making better future decisions and preserving knowledge and expertise within a company.

2.1 Research Challenges

RC1) How to model (i.e., describe and represent) the requirements and constraints that are inputs to the generative secure design approach. The important factors are to provide an appropriately expressive modeling solution, with the necessary level of underlying formality, which also considers usability for designers.

RC2) How to model the secure design alternatives generated by the approach. Similarly, we must extend and/or develop several modeling languages which capture secure design at an appropriate level of technical detail, while being mappable to the problem domain.

RC3) How to explore the solution space for the security problem presented by the user and generate secure design alternatives. The key challenge is to develop genetic algorithms and optimization techniques to generate multiple design alternatives that are sufficiently varied, possibly creative (i.e., non-trivial, surprising) and yet relevant to the criteria captured in RC1.

RC4) How to present and explain the design alternatives to the user. The key challenge is the understandability of the alternatives and, most importantly, providing an explanation to the designer of the key, qualifying differences among the alternatives, in order to enable an informed choice.

RC5) How to capture the rationale behind the user's selection from design alternatives, and how to feed back this rationale into the iterative process so that new alternatives can be generated. The rationale should be recorded as preference learnt from the user in a form compatible with the inputs modeled by RC1. These preferences should guide the generation of further secure designs using the techniques of RC3.

RC6) Local and global evolution of the generative algorithms. Preference knowledge can be reused in future designs across multiple instances of the tool, e.g., across teams, departments, or even globally (as a service provided across companies). Clearly, this reuse brings considerations concerning intellectual property and strategic advantages, which need to be addressed.

3 RESEARCH DIRECTIONS

Solving the challenges listed in Section 2.1 requires a multidisciplinary approach that involves expertise in requirements engineering (specification of the security design goals), software design (modeling of the secure design solutions), search-based software engineering (generation of alternatives via genetic algorithms and optimization), as well as knowledge representation (to capture design explanations and record designer preferences) and learning (to improve the generation according to the choices of the designer). For the first time, all these competences are discussed jointly in the domain of cyber security. We envision that the successful development of the Generative Secure Design technique requires novel research is required in three areas: modeling, generation, and learning. Each of these aspects is discussed in the following sub-sections.

3.1 Modeling

This area relates to RC1, how to model the problem, and RC2, how to model the secure design solutions. We must utilize and build upon existing languages with both a visual and textual modeling component in order to allow users to express security requirements, constraints, assumptions, and other key elements of the problem domain. Typically, users will be improving, changing, or extending the security of existing systems. As such, we must allow users to express constraints and information concerning the current system design using modeling notations currently used in industry (e.g.,

UML, SysML, Simulink, or Architecture Description Languages such as EAST-ADL). The key challenge in this work package is finding an effective balance between expressivity and usability.

The modeling solution needs to be validated by building a realistic library of security design templates (e.g., starting from security patterns and reference security architectures). Such a library is used as a seed by the generative algorithms, as described in Section 3.2.

How to get there. This research area should build upon extensive work in the fields of security requirements modeling [3], focusing on usability, expressiveness and the potential to map or transform these models into the aforementioned architecture and design models. However, there is a need to evaluate existing security-oriented requirements and design languages, focusing particularly on the concepts needed to express security concerns in design and architecture, and their link to security requirements. This investigation should identify appropriate formal underpinnings, building on existing work, keeping in mind algorithms and techniques particularly suited for generative design. Given the ultra-large scale nature of our challenge, this research should also focus on creating a modeling solution with many complementary interconnected views at different levels of abstraction, e.g., a security requirements view, a quality view, a design constraint view, a behavioral view, a security solution view, etc. Although we expect the problem and design spaces to be represented via a variety of model types (e.g., requirements models, UML, ADLs), the languages must be made compatible, and the connections between models must be understandable to users.

3.2 Generation

This research area relates to RC3, how to generate secure design alternatives, and RC4, how to explain the design alternatives to the user. Given the set of models that describe the requirements as well as an existing design, the goal is to apply evolutionary search to generate a set of augmented designs with improved security properties and present them to the designer. The modeling languages described in Section 3.1 provide a space to explore in which the seed design provided by the designers is combined with secure design templates to produce augmented designs. This research area should detail algorithms for combining these different, fragmented design elements into unified designs as well as define metrics and evaluation methods to characterize the security properties in a form that can be understood by designers. A central goal of the presentation is to show both the design enhancements for security and the properties they entail. The state-of-the-art has highlighted the two central problems in interactive search-based software engineering: (a) how to find diverse solution alternatives of high quality and (b) how to visualize and explain the complex characteristics of the proposed solutions to the software engineer in manner that allows them to guide and steer the process of search and exploration.

How to get there. The core solution strategy is a multi-objective evolutionary search that combines elements from the library of secure design templates with the seed designs to improve security properties. Objectives to be optimized are of three main types: general quality metrics, security specific properties, and diversity of the augmented design relative to previous designs. While the first two are obviously important, the diversity objectives ensure that only sufficiently novel solutions are presented to designers for feedback. Finding algorithms to trade-off between diversity and the more traditional quality criteria is an important challenge.

Once a small set of promising design candidates have been found they must be presented in a sensible way for the designer. It is desirable to use multiple linked model views in combination with visualizations of the different objectives and security properties. The design view would focus on showing the difference between the proposed candidate and the original, seed design, while objectives and properties would be shown using more traditional data visualization approaches such as linked scatterplots and "small multiples". The challenge is to efficiently utilize screen space to show the subset of security properties that the designer is currently focused on, while allowing efficient switching between such sets. The explanations offered by the linked presentations need to be enhanced with alternative, textual summaries that describes the main design enhancements and their effect on the design.

3.3 Learning

At each iteration of the design process, the user indicates the designs that they prefer from among a set of augmented designs generated by the algorithms of Section 3.2. Accordingly, this research area addresses the challenges of how to capture and learn from the user's rationale for these selections. The rationale must be captured in a form that enables online learning during the current design process: the information is fed back to algorithms so that designs generated in subsequent iterations incorporate the user's preferences (RC5). The rationale information gathered from the generative design tool must also support offline learning: in other words, it must be capable of being transformed into secure design knowledge that may be shared with other users of the design tool within the same project and company, or across companies (RC6).

How to get there. The rationale for the user's preferences could be their knowledge of the system being developed, in which case their preferences may be captured as additional or refined requirements, and strengthened or weakened constraints. Alternatively, the rationale could be the user's knowledge of the relative importance placed on security-related qualities by the organization, and trade-offs that are possible with other design qualities. This rationale may be captured as changes to the metrics that are used by the generative algorithms to identify the best augmented designs.

To address RC5, there is a need to investigate effective methods to a) capture changes arising from user rationale into the requirements model, and b) feed the changes into the generative design algorithm and its optimization metrics. In this respect, it is possible to build upon existing research on search-based algorithms in which the problem (here, the requirements models) or the objectives (here, optimizing security-related qualities of the design) change between iterations of the search process, such as existing research on interactive search-based generation of test data [13]. In addition, it should be investigated how to capture rationale that will not result in a direct change to the requirements nor optimization objectives, but instead provides argumentation to support the decisions during the design process.

To address RC6, there is a need to investigate machine learning techniques to extract knowledge from the combination of rationale

information, requirements, and the resulting secure design. For example, the knowledge may be represented in the form of project- or company-specific design patterns that supplement the library provided in WP1, for which the rationale information identifies the contexts in which it will be beneficial to apply a pattern. A particular challenge is to identify knowledge that is derived from proprietary or strategic information (like vulnerability data [14]) so that the dissemination of such patterns may be controlled appropriately.

4 STATE OF THE ART

The most related work is [6]. Our vision goes beyond code refactoring and focuses on generative design, while also including the novel aspects of user interaction and preference learning.

Search-based software engineering. In the last 15 years, there has been a growing use of search and optimization to support software engineers, an approach termed search-based software engineering (SBSE) [8]. Only recently, evolutionary search has been explored in more depth and, for example, was proposed as a generative design methodology for object-oriented class designs that were optimized for low coupling and high cohesion [15]. Search-based approaches have also been demonstrated as effective in generating designs that satisfy requirements expressed as goal models, while exploring the trade-off between functional and non-functional objectives [2], although we note that this approach did not have the ability to incorporate human guidance. An interactive search-based approach has been suggested as a way to refactor a code base in order to comply with an intended architecture [12]. As mentioned earlier in the paper, the state-of-the-art has highlighted a central problems in interactive search-based software engineering, i.e., how to find diverse solution alternatives of high quality. The generation of diverse alternatives critically depends on the availability of a suitable metric to quantify the similarity between proposed solutions. Early work [5] showed how constructs from information theory can be used as general diversity metrics in SBSE, and recently it was shown [4] how these metrics can be generalized to find diverse sets of test cases with high fault-detecting ability.

In summary, even though search and optimization has been proposed and explored in many areas of software engineering, there is a lack of applications for design refinement in general, and secure design generation and refinement in particular.

Requirements, design and security. Related work addresses the selection of security and privacy mechanisms, like security design patterns, to satisfy a security objective or a specific security requirement [11, 19]. The mentioned work provides decision support (e.g., in the form of tables) for the designer to assess whether a specific security design pattern is fit for the problem at hand. This paper sets forward the more ambitious aim of automating the transition from the problem domain to the solution space. An analysis of the work devoted to transforming or mapping requirements to downstream artifacts, such as UML models, has shown that such work often provides partial solutions, since mapping non-functional concerns to concrete architectural elements remains challenging [9]. A few requirements engineering approaches have begun to make use of iteration and learning. In some approaches, users are asked to iteratively select and evaluate requirements knowledge generated by model checking over goal modeling, leading to refined functional requirements [1]. Other work incorporated iterative user judgment into the qualitative goal model analysis procedure for early requirements analysis [10]. Specific to security, when developing satisfaction arguments for security requirements, some approaches iteratively ask designers for additional design information when arguments fail [7].

In summary, no work exists which generates candidate security designs from requirements knowledge and iteratively collects user feedback concerning the applicability of such designs.

ACKNOWLEDGMENTS

In memory of Simon Poulding, who contributed to the inception of this work.

REFERENCES

[1] Dalal Alrajeh, Jeff Kramer, Alessandra Russo, and Sebastin Uchitel. 2009. Learning Operational Requirements from Goal Models. In *Proceedings of the 31st International Conference on Software Engineering (ICSE'09)*.
[2] Frank R. Burton, Richard F. Paige, Simon Poulding, and Simon Smith. 2014. System of Systems Acquisition Trade-offs. *Procedia Computer Science* 28, Supplement C (2014), 11 – 18.
[3] Fabiano Dalpiaz, Elda Paja, and Paolo Giorgini. 2016. *Security requirements engineering: designing secure socio-technical systems.* MIT Press.
[4] Robert Feldt, Simon Poulding, David Clark, and Shin Yoo. 2016. Test Set Diameter: Quantifying the diversity of sets of test cases. In *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST'16)*.
[5] Robert Feldt, Richard Torkar, Tony Gorschek, and Wasif Afzal. 2008. Searching for Cognitively Diverse Tests: Towards Universal Test Diversity Metrics. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08)*.
[6] Shadi Ghaith and Mel Ó Cinnéide. 2012. Improving Software Security Using Search-based Refactoring. In *Proceedings of the 4th International Conference on Search Based Software Engineering (SSBSE'12)*.
[7] Charles Haley, Robin Laney, Jonathan Moffett, and Bashar Nuseibeh. 2008. Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Trans. Softw. Eng.* 34, 1 (Jan. 2008), 133–153.
[8] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Comput. Surv.* 45, 1 (Dec. 2012), 1–61.
[9] Jennifer Horkoff, Tong Li, Feng-Lin Li, Mattia Salnitri, Evellin Cardoso, Paolo Giorgini, and John Mylopoulos. 2015. Using goal models downstream: A systematic roadmap and literature review. *Requir. Eng.* 6, 2 (2015).
[10] Jennifer Horkoff and Eric Yu. 2016. Interactive Goal Model Analysis for Early Requirements Engineering. *Requir. Eng.* 21, 1 (March 2016), 29–61.
[11] Tong Li, Jennifer Horkoff, and John Mylopoulos. 2014. Integrating Security Patterns with Security Requirements Analysis Using Contextual Goal Models. In *Proceedings of the 7th IFIP WG 8.1 Working ConferenceThe Practice of Enterprise Modeling (PoEM'14)*.
[12] Yun Lin, Xin Peng, Yuanfang Cai, Danny Dig, Diwen Zheng, and Wenyun Zhao. 2016. Interactive and Guided Architectural Refactoring with Search-Based Recommendation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16)*.
[13] Bogdan Marculescu, Robert Feldt, Richard Torkar, and Simon Poulding. 2015. An Initial Industrial Evaluation of Interactive Search-based Testing for Embedded Software. *Appl. Soft Comput.* 29, C (April 2015), 26–39.
[14] Riccardo Scandariato and James Walden. 2012. Predicting vulnerable classes in an Android application. In *Proceedings of the 4th international workshop on Security measurements and metrics (MetriSec'12)*.
[15] Christopher L. Simons, Ian C. Parmee, and Rhys Gwynllyw. 2010. Interactive, Evolutionary Search in Upstream Object-Oriented Class Design. *IEEE Trans. Softw. Eng.* 36, 6 (Nov. 2010), 798–816.
[16] Kim Wuyts, Riccardo Scandariato, and Wouter Joosen. 2014. Empirical evaluation of a privacy-focused threat modeling methodology. *Journal of Systems and Software* 96 (2014), 122–138.
[17] Koen Yskout, Riccardo Scandariato, and Wouter Joosen. 2012. Does Organizing Security Patterns Focus Architectural Choices?. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*.
[18] Koen Yskout, Riccardo Scandariato, and Wouter Joosen. 2014. Change patterns. *Software & Systems Modeling* 13, 2 (May 2014), 625–648.
[19] Koen Yskout, Riccardo Scandariato, Bart De Win, and Wouter Joosen. 2008. Transforming Security Requirements into Architecture. In *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security (ARES'08)*.