The Trouble With Security Requirements

Sven Türpe
Fraunhofer Institute for Secure Information Technology SIT
Darmstadt, Germany
Email: sven.tuerpe@sit.fraunhofer.de

***Abstract*—Manifold approaches to security requirements engineering have been proposed, yet there is no consensus how to elicit, analyze, or express security needs. This perspective paper systematizes the problem space of security requirements engineering. Security needs result from the interplay of three dimensions: threats, security goals, and system design. Elementary statements can be made in each dimension, but such one-dimensional requirements remain partial and insufficient. To understand security needs, one has to analyze their interaction. Distinct analysis tasks arise for each pair of dimensions and are supported by different techniques: risk analysis, as in CORAS, between threats and security goals; security design, as exemplified by the framework of Haley et al., between goals and design; and security design analysis, such as Microsoft's threat modeling technique with data flow diagrams and STRIDE, between design and threats. All three perspectives are necessary to develop secure systems. Security requirements engineering must iterate through them, because threats determine the relevance of security goals, security design seeks ways to fulfill them, and design choices themselves influence threats and security goals.**

***Index Terms*—Computer security, requirements engineering, system analysis and design, security risk, threat model, vulnerability, information security, software design, solution design**

I. Introduction

Security requirements define the difference between a functional but vulnerable system and one that can withstand malicious interference and abuse in its operational environment. Regardless of their functional correctness and their fulfillment of other requirements, software and systems are by default vulnerable to attacks. To make them less vulnerable, the development process needs to shape their security properties by adding security mechanisms and implementing them correctly, by observing security principles, and by avoiding defects attackers could exploit. Security requirements guide this design, implementation, and verification work.

The literature describes a smorgasbord of approaches, techniques, and notations for security requirements engineering and related tasks [1]–[5]. Alas, the various proposals can apparently not even agree on a common definition what a security requirement is [3], let alone a common general approach. Moreover, industry practitioners have developed their own set of tools and techniques to identify security concerns early during development [6]–[10].

Security requirements are often considered non-functional [11] and one can even find complaints about functional expressions of security requirements in terms of security mechanisms [12]. Empirical evidence suggests that non-functional requirements tend to be discovered by software architects rather than being specified by stakeholders [13]. Nevertheless, some proposed approaches treat security requirements as something to be elicited from stakeholders and passed to architects and designers. Techniques grown in the software industry [6], [8], on the other hand, are focused on design and design review by developers.

This paper aims to connect some of the dots and proposes a metamodel of security requirements engineering. Instead of a fine-grained conceptual model of security, as applied in some of the literature [1], [14], we use a broader brush here and argue that security needs have three essential dimensions. The first is threats, as security needs do not arise in benign environments where no actors with malicious intent exist. The second is stakeholder goals. Threats imply security needs only inasmuch as their consequences collide with the goals of a system's stakeholders. The third dimension is the design of the system, particularly its security design. It shapes the system's security properties and vulnerabilities, which determine how threats can or cannot cause undesired consequences.

Statements concerning security needs can be made in each dimension independent of the others, but analysis is necessary to validate such statements, to derive design choices, and to verify development results. With threats and goals one can reason about risks—possible and likely consequences of unmitigated threats for the goals of a system and its stakeholders. CORAS [15] is an example of a risk analysis technique. Between goals and design we find security requirements engineering in a narrower sense, aiming to prepare and verify design decisions. An example is the security requirements engineering framework by Haley et al. [16]. Finally, between design and threats one reasons about possible interactions between adversaries and a system. Microsoft's threat modeling technique with data flow diagrams (DFD) and STRIDE [6] represents this type of analysis.

All three types of analyses are necessary to understand security needs and build secure systems. The requirements engineering and development process has to iterate through them. This is in part a consequence of the Twin Peaks model [17] and in part one of adaptive adversaries reacting to design choices. Instead of merely adapting the Twin Peaks model for security [18], it should therefore be extended to include threat models as a third peak.

Regardless of its focus and approach, security requirements engineering entails some epistemic challenges. First, adversaries influence the success or failure of a security design, but other than legitimate stakeholders they are not available

for requirements engineering activities. Second, security itself is complex, and security requirements techniques have to deal with this complexity. Third, reasoning about security, as required in the three requirements analysis tasks, is hard. Future research might take on these challenges.

II. Security as a Design Objective

Security is not simply a set of features or a functional component to be added to a system. Security is rather a cross-cutting concern to be considered throughout development. Security needs of a system arise from the expected presence of threats in its operational environment: *"Security engineering is about building systems to remain dependable in the face of malice, error or mischance"* [19]. Threats can be intentional ("malice") or unintentional ("error or mischance"). In a narrower sense, security deals primarily with intentional threats caused by intelligent adversaries, which actively pursue their respective agenda and are harder to fend off than unintentional accidents.

A. From Policy Enforcement to Security Development

In its early days, computer security aimed to solve a straightforward problem in a simple setting: to enforce organizational security policies in isolated computers belonging to the organization [20]. Attention was focused on operating systems and the question how to enforce access control policies there [21]–[25], so that they would apply to all application programs. A set of security design principles [26] soon emerged for this setting and a large body of literature on access control and security policies has grown ever since. In parallel, the old art of cryptography gained wider attention and entered a new era with the invention of public-key crypto systems [27]–[29].

While access control and cryptography continue to be cornerstones of computer security, information technology has outgrown early-day assumptions and some—but not all—of the early principles [30]. On the one hand, having a single centralized access control mechanism is no longer feasible with vast numbers of computers connected across organizational and other boundaries. On the other hand, exploitable software defects, such as memory corruption bugs [31], [32] allowing attackers to inject their own code into a running program, turned out as a frequent problem [33], [34]. Attacks often exploit such vulnerabilities or weaknesses caused by system management or user behavior, rather than flaws in security mechanisms themselves [35], [36]. With the growing number, diversity, complexity, and interconnection of computer systems the number and diversity of adversaries and their possible attacks has also grown.

B. Security Design

Consequently, security comprises besides functional requirements also quality aspects [11], [37]. Security functions become ineffective if they themselves, or components they depend on, have implementation or design flaws or if the security architecture of a system permits their circumvention. To address potential vulnerabilities, development has to take care of:

- *security design patterns* [38]–[40] and architectural properties like the attack surface [41] or defense in depth [42],
- *dependencies between security functions,* such as the reliance of access control on authentication and of remote authentication on network security,
- *technology-specific flaws and defects* like those collected in the Common Weakness Enumeration (CWE) [43] and the OWASP Application Security Verification Standard [9],
- *vulnerabilities inherited from used components,* as in the case of the notorious *Heartbleed* vulnerability in OpenSSL [44],
- *human factors* and usability issues [45], [46] affecting the strength of security features, and
- *possible abuses of required functions* by otherwise legitimate users or by outsiders.

These and other security considerations have to be made in addition to any development work addressing primary requirements. Systems are usually created to serve a purpose and then they should also be secure. This makes security a *secondary* concern. Specific security concerns are *dependent and implied* by other requirements or by design decisions. Security as a whole is a *cross-cutting* topic that cannot be isolated in a particular component, and security concerns are *non-uniformly* distributed across the parts of a system.

C. "Secure" Means "Not Vulnerable"

Positive definitions of security are difficult if not impossible [37]. Security is about preventing attacks by adaptive adversaries, who actively seek routes to success [47] and ignore rules that are not being enforced by effective mechanisms [48]. The secondary, dependent nature of security concerns adds to the challenge: Similar design features can have vastly different consequences in different contexts.

Defined negatively, security development aims to prevent vulnerabilities in development results. *"A vulnerability is a weakness in the system that an attack exploits"* [49], in other words, a feature or property of a system that helps an attack to succeed. The more vulnerable a system is, the higher is the expected success rate of attempted attacks [14] and the more choices an attacker has. For substantial security improvement it does not suffice to mitigate a random collection of vulnerabilities. Considering attack adaptation, one has to care about classes of vulnerabilities that are equivalent from an attacker's point of view as they afford comparable results with comparable effort and prerequisites. However, it would on the other hand also be inappropriate to overspend on security and address vulnerabilities that could theoretically be exploited by some attacker but are unlikely to matter in practice.

A further challenge lies in the nature of vulnerabilities as design features (or defects) of a system. Vulnerabilities often exist in side effect behavior that does not affect normal use but is exploitable in an attack [50]. For example, weak or missing encryption of data in transit over untrusted networks

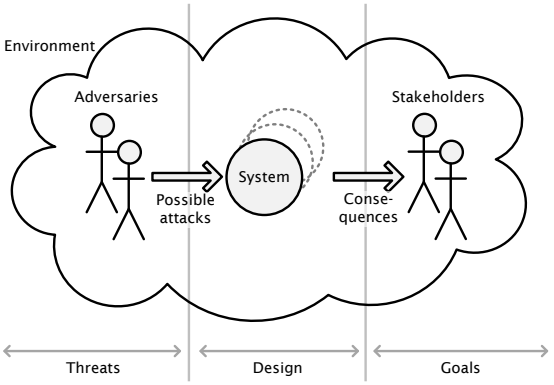

Figure 1. General model of security problems. Adversaries may attack a system, causing consequences for its stakeholders.

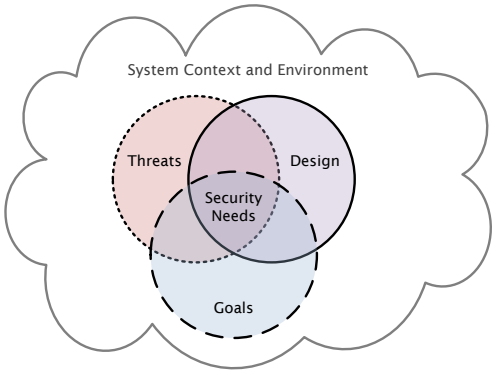

Figure 2. Dimensions of security needs. Threats, security goals, and the design of a system together determine security needs.

has not the slightest impact on a system's function, reliability, or performance, but it allows an attacker to obtain information by sniffing network traffic. This has two consequences: First, software and systems are vulnerable by default and it takes dedicated effort to make them less so. Second, unlike defects that affect users in some way, vulnerabilities tend to remain invisible until one specifically looks for them.

III. Three Dimensions of Security

A. Security Needs

While a general desire for security is easy to express, development needs more precise guidance. To provide this guidance is the purpose of security requirements. A requirement is a *"statement which translates or expresses a need and its associated constraints and conditions"* [51]. Requirements express needs in such a form that the design and development process can fulfill them and its results can be verified. Security requirements do this for security needs. As with other requirements, stakeholder goals are their root.

Security involves another type of actor besides stakeholders—adversaries. An adversary has goals like a stakeholder and will pursue them in or through a system, but those goals and the adversary's success are not to be supported. Adversaries are unwelcome stakeholders whose success would run counter to the goals of the legitimate ones, hence the notions of anti-goals and anti-requirements [52], [53]. Fig. 1 illustrates the adversarial relationship that leads to security needs: Adversaries may attack a system, causing consequences that affect legitimate stakeholders[1].

Let us condense this general model into two definitions:

Definition (Security Needs)**.** *Security needs of a system or software product are design objectives concerning the protection of its stakeholders from consequences of (intentional) threats that conflict with the stakeholders' goals.*

[1]Note that the separation of stakeholders and adversaries is an idealization, as rogue users or insider threats blur the line between them. However, for any particular attack it should be possible, at least in principle, to distinguish the perpetrators from their victims.

Definition (Security Requirements)**.** *Security requirements express security needs in a form suitable to inform security design and make its results verifiable.*

The distinction between security needs and security requirements aims to evade preconceptions about the latter [54].

Not explicitly included in these definitions is the environment in which a system operates. Environmental conditions modify security problems and thus security needs. For example, different systems may be exposed to different adversaries, the presence or absence of alternative targets may influence adversary behavior, and attacks may have consequences outside a system. As Courtney's first law puts it, *"You cannot say anything interesting (i.e., significant) about the security of a system except in the context of a particular application and environment."* [55].

B. Threats, Goals, and Security Design

Three elements interact to create a security problem: *threats* to which a system is exposed, the *(security) design* of the system, which is shaped during development, and stakeholder goals, with which the consequences of manifest threats—i.e., attacks—collide. The likelihood and consequences of attacks depend on adversaries and system properties; which consequences matter depends on the stakeholders' security goals. Therefore we need to consider three dimensions as shown in Fig. 2 and outlined below to discuss the security needs of a system.

1) Threats: Threats are the *raison d'être* of security needs. Threats are an objective, independent factor a system is confronted with. Threats can only be understood and addressed, not changed or negotiated. Threats as such are latent and merely a potential [14]; how this potential translates into actual attacks depends also on the target(s) and the environment. Adversaries (a.k.a. threat agents) as the source of security threats are intelligent, adaptive actors pursuing their own agenda and disregarding damage inflicted on others. A threat therefore corresponds with a range of possible attacks that are viable and promising for the respective adversaries. Threats

may target a system for its own sake or as an instrument to achieve goals elsewhere.

2) Goals: Stakeholders have explicit or implicit security goals. The operation and use of a system is for them a means to some end. In a benign environment that would be it, but actual or feared threats impose security goals. Perhaps the most general formulation of a security goal is to let a system operate as intended without unbearable disruptions or side effects, regardless of any threats. More specific goals follow by refinement. Security goals are traditionally often expressed in terms of assets and properties like confidentiality, integrity, or availability to be preserved, but may also concern, for example, conditions for collaboration [56] or economic boundaries. Security goals can exist without actual threats, but defense against imaginary threats would be an investment without return. Goals can therefore be negotiated and traded off and any particular goal may be more or less reasonable to pursue given the cost of doing so.

3) Design: Security design has a twofold meaning, referring on the one hand to the features and properties of a system that determine its security and on the other hand to efforts to shape these features and properties. A system's security design influences how this system can be attacked and what the consequences are. Security design is intertwined with other design aspects like functionality, architecture, and usability. Since vulnerabilities often exist in side effect behavior, security design takes place inadvertently unless one makes a conscious effort to shape the security properties of a system. Even then, any further design or implementation decision can change security properties. As systems are facing multiple adversaries and attacks and stakeholders may pursue diverging goals, security design involves tradeoffs.

C. Example: Bicycle Theft

As an example illustrating the three dimensions of security needs, consider bicycle theft [57]. The *threat* of theft results from the existence of thieves—adversaries who disregard others' ownership rights in favor of their own gain. This threat concerns all movable objects, but its intensity varies with exposure, value, ease of taking away, and other factors. The theft threat conflicts with the owner's *goals* to retain ownership until voluntarily transferring it and to use the bicycle at any time. The *design* of a bicycle as a lightweight human-powered vehicle makes it easy to move. Locking is a common design feature to counter the theft threat. Thieves may adapt by choosing different targets, by attempting to break locks, or by stealing bicycle parts.

D. Common Concepts, Inconsistent Terminology

The trinity of concepts in Fig. 2 appears, in varying wording, all over the security literature. Table I shows some examples. These materials take different perspectives, which their respective terminology reflects. For example, Willis [59] focuses on attacks, whereas the Common Criteria [60] are a framework for product security certification. The former therefore speaks of vulnerabilities and attack consequences and the latter of countermeasures and assets. More importantly, definitions of security [37, p. 316] or security engineering [19, p. 3] use expressions like *malice* (threat), *dependable* (goal), and *building systems* (design) that map straightforwardly to these concepts.

The literature proposes several finer-grained ontologies of security concepts [1], [60]–[63]. The three dimensions can be found there as well, although broken down into further pieces. For example, the Common Criteria's general model [60, part 1, sec. 7.1] posits six concepts: *owners* and *assets* (goals), *threat agents* and *threats* (threats), *countermeasures* (design), and *risk* as something determined by the others.

IV. One-Dimensional Security Requirements

Elementary security requirements can be expressed in each of the three dimensions without much regard for the others. However, to validate such statements about security needs or to translate them into design decisions one needs to make assumptions about the remaining dimensions. These assumptions often remain implicit, but sometimes become visible in comparison between different systems or contexts. A mix of assertions across the three dimensions is the likely result of naive security requirements elicitation.

A. Security Goals

Security goals describe the desired protection of a system and its environment. They capture conditions to be achieved regardless of threats and design consequences. Goals may describe, for example:

- assets and their protection needs: "Documents concerning inventions must remain confidential until a patent application has been filed,"
- dreaded consequences: "Large-scale leakage of personal data will cost us reputation and customers,"
- organizational policies pertinent to the information processed in a system: "Only staff with sufficient security clearance can access classified information," or
- legal or business requirements: "The system must comply with European data protection law."

The first two items describe protection goals, the other two are formal compliance goals originating from assumed protection needs.

Such goals often pre-exist when development begins and can be elicited as domain knowledge of stakeholders. However, assets may be missed or their protection needs may be over- or understated. When considering only goals, it is easy to request protection of everything against all evils, but the cost of protection requires prioritization and therefore some understanding of the threats faced.

Some security goals can be refined into functional security requirements. However, security design is underspecified by only functional requirements: the implementation of required security functions alone does not guarantee the security need to be satisfied with respect to actual adversaries. Nonetheless, security functions can provide a useful protection baseline and their presence may also limit liability in the case of an incident.

Table I
VARYING TERMINOLOGY FOR THE THREE DIMENSIONS OF SECURITY NEEDS

Dimension	**Shostack** [8, p. 219]	**Landwehr** [58]	**Willis** [59]	**Anderson** [19, p. 3]	**Goertzel et al.** [37, p. 316]	**Common Criteria** [60, pt. 1, 7.1]
Threats	Threats	Adversary, threat	Threat to a target	Malice, error, mischief	Threats	Threat agents and threats
Goals	Requirements	Policy	Consequences	Remain dependable	Preserve dependability	Owners and assets
Design	Mitigations	Mechanism	Vulnerability	Building systems	Ability . . . to protect itself from sponsored faults	Countermeasures
Security needs, requirements	— (Security)	Assurance	Risk	Security engineering	Protection against disclosure, subversion, or sabotage	Risk (reduction)

Security goals are useful as a reference in verification tasks like penetration testing. Formal goals may also imply defined evaluation procedures. Effective protection of assets, on the other hand, is difficult to demonstrate. If a possible vulnerability has been found, its impact can be evaluated against known goals. Prioritization is difficult without an understanding of threats.

B. Threats

A threat statement describes adversaries—their goals, capabilities, or expected behavior. Threats can be characterized without reference to specific security goals or system design features, for example in terms of:

- adversary capabilities: "A nation-state actor with considerable resources," or "an adversary controlling a botnet on the Internet,"
- objectives and success conditions: "Cyber criminals work for profit and aim to evade apprehension,"
- target selection behavior: "opportunistic," "targeted," "scalable,"
- modus operandi: "Intelligence services employ 'evil maid' attacks, wherein they inspect and possibly tamper with unattended personal mobile devices," or
- technical attack patterns: "Transparent rerouting of network traffic to run through a host under the control of the attacker."

Security needs follow from threats if a system is exposed to them, adversaries are interested in it as a target, and the resulting risks are not acceptable.

Eliciting complete and reliable threat models from stakeholders is hard; even security experts struggle with this task. Validation is equally difficult. The challenge is to distinguish real threats from unsubstantiated fears [64] while simultaneously avoiding too restrictive assumptions about what adversaries might want and do. Neither ignorance nor excessive paranoia lead to useful threat models.

Threat models motivate security needs but do not define them to the extent that one could derive a security design specification from threat descriptions alone. They provide therefore a rather indirect design guidance, a frame of reference to evaluate design choices and features in. Threat models serve as a useful input into verification tasks like penetration testing. However, the effectiveness of a security design against real threats remains difficult to assess due to adaptation of attacks and many unknown parameters. Technical attack patterns may be easier to work with than adversary models, particularly when it comes to baseline protection against common vulnerability patterns, but such patterns can be meaningfully discussed only in relation to a system design.

C. Security Design

Finally, security needs can be described by statements concerning the design of a system or software product itself. Such statements are focused on the solution space and omit or merely imply their justification. One can state, for example:

- required security functions: "The system must log an audit trail of security events,"
- design detail with security implications: "Session IDs must be long, random, and unique," or
- common vulnerabilities to avoid: "The web UI must not contain cross-site scripting vulnerabilities."

Expressing security needs in terms of the design space may seem like a shortcut, but there can be good reasons to jump right to design. First, some security functions may be justified by business requirements and customer preferences [65], rather than by threat and security goal considerations. Second, some security functions—e.g., user authentication, access control, and encrypted communication—are common across a wide range of systems and applications as they solve common problems. Third, baseline protection against common vulnerabilities is almost always needed. Fourth, some technology and architectural choices entail specific security design requirements. While following best-practice guides may not suffice to create a secure system, it is often a necessary condition.

Elicitation of security design requirements from stakeholders can yield partial and skewed results. Security experts and developers may be better suited to select a reasonable set of design guidelines, but even they can fall victim to availability heuristics and other biases. Design guidance by stakeholders needs to be validated against goals and threats

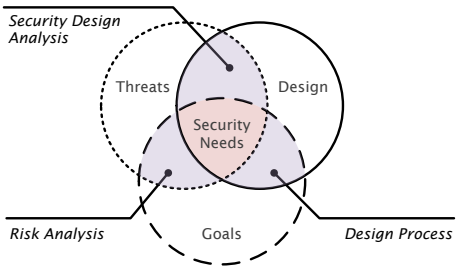

Figure 3. Analysis tasks. Each pair of dimensions leads to a distinct perspective.

to prioritize it and also to find gaps due to overestimation of the protection afforded by security mechanisms. Design guidance straightforwardly translates into specifications and is straightforward to verify later on.

V. Analysis Tasks

Single-dimension statements need validation and then translation or refinement into applicable design guidance. This is ultimately a three-variable problem, where any change in one dimension may entail changes or new questions in the remaining two. Changing goals imply changing design objectives and threat prioritization, changing threats entail different security goals and change the success criteria for design choices, and design changes may lead to new goals and to threat adaptation. This makes it difficult, if not impossible, to refine a set initial requirements into final design guidance in a the single pass of a waterfall approach.

To mitigate this difficulty, one may keep one dimension temporarily fixed while focusing on interdependencies between the remaining two. As we will see in the next section, this is indeed what some security requirements techniques do. This yields the three analysis tasks indicated in Fig. 3, one for each pair of dimensions. Risk analysis aligns security goals with threat models and prioritizes them. The design process translates security goals into design decisions. Security design analysis looks at the security properties that result from design decisions in relation to threat models.

A. Threats and Goals: Risk Analysis

Risk analysis pairs security goals with opposing threats, refines either component as necessary, and ranks goal-threat combinations by goal importance and expected damages. Risk analysis takes an abstract and "what-if" point of view. Technology and design considerations play a minor role and effects of design choices are often merely estimated or assumed. On the one hand is the impact of design choices on the overall risk profile hard to estimate. On the other hand are technical details irrelevant for many scenarios. If, for example, the goal at hand is to keep certain data confidential against a variety of leakage threats, the precise technical way of leaking is less important for risk analysis than the frequency and scale of possible incidents. Once a risk model has been established, there may also be alternative approaches to risk mitigation, avoidance, or transfer, which likely rely on different features of the system. Risk analysis leads to a better understanding of the security problem and of general solution strategies, but does not detail the solution.

Consider the bicycle theft example from III-C above. Going back and forth between threat models and security goals, we may refine our understanding of the problem as follows: The utility and the value of a bicycle may be lost temporarily, e.g., if damaged and repaired or if stolen but recovered, or permanently. Permanent loss can be considered the maximum-damage case; the average bicycle is worth a few hundred euros. Thieves may target this value and try to sell stolen bicycles, be content with the lower scrap value, or take a bicycle solely to ride it themselves. In the light of this threat model our general security goal translates into a combination of strategies: (1) increase the immediate cost of stealing, (2) make the loot less valuable for the thief, (3) ease recovery, and (4) evade targeting.

B. Goals and Design: Design Process

The design process of a system translates security goals into design choices. High-level design takes place in a process of exploration, evaluation, and decision-making [66]. Security is one of many facets of design, a particular perspective from which to consider and select options. Security goals guide the selection of security mechanisms, the definition of policies and security models, and the application of security principles and patterns.

Security design is not a straightforward, top-down refinement of security goals. Even if a security goal is closely aligned with the capabilities of a security mechanism, such as confidentiality with encryption, mere deployment of the mechanism does not imply fulfillment. The scope of protection of a security function is usually limited, so that multiple functions must be combined for effective protection throughout the lifecycle of some entity. The efficacy of security mechanisms also depends on their place in the security architecture, mechanism implementation, and further factors. Security goals therefore cannot determine security design, they can only guide it.

In our bicycle theft example we can choose from a variety of locks and ways of locking to make a bicycle harder to steal. Common lock designs are small and light enough for easy carry. Some variation exists in their design, mode of operation, key management, and strength. Locks can be applied in different ways: through parts of the bicycle (e.g., wheels, frame) or through bicycle parts and through or around objects in the environment, like lamp posts or bike racks. The thief's cost of breaking the lock depends on the lock; the necessity to do so as a prerequisite for stealing depends on the way the lock is being applied. As an additional security mechanism one might mark the bicycle with a unique ID and register it in a database, which makes it harder to sell without proof of ownership and easier to recover.

C. Design and Threats: Security Design Analysis

To understand how to fulfill a system's security goals, one has to analyze how its threat environment may behave in the presence of the system, how adversaries may tamper with it, and how the system behaves under attack. This assessment takes place in security design analysis. The perspective is similar as in penetration testing [67]–[69], but security design analysis does not wait until after implementation. Like penetration testing, security design analysis is exploratory.

Security design analysis looks for possible unmitigated vulnerabilities that allow attacks to succeed. Although attackers cheat and break rules [48], their behavior often follows patterns and stereotypes. By interpreting these patterns for a specific system design in what-if scenarios, one can generate lists of attacks and attack variants this system may encounter, evaluate their consequences, and devise mitigations if necessary. Besides attack patterns, security design analysis may also cover architectural properties like the attack surface.

Examples for collections of attack patterns are STRIDE and CAPEC. The acronym STRIDE captures six common attack tactics: Spoofing, Tampering, Information Disclosure, Denial of service, and Elevation of privilege [6]. The CAPEC database [70] contains a larger number of more detailed attack patterns, many of which correspond with certain vulnerability types. As a result of pondering attack tactics, one may find issues outside the scope of the established security goals.

In the bicycle theft example, the relevant attack patterns differ from those applicable in computer security. Nevertheless we can look at any security design variant and analyze what a thief with certain objectives and capabilities may do to it. For example, any lock at all may be good enough to fend off those merely looking for a joyride. But a professional bicycle thief may bring tools to the scene to crack locks, come with a car and load bicycles that cannot roll but are not attached to other objects, or resort to dismantling and stealing valuable components rather than entire bicycles.

VI. Analysis Techniques

The three analysis tasks correspond with different types of representations and techniques. As the purpose of this paper is not to survey the literature on security requirements engineering—several surveys exist [1]–[4]—we pick here one representative for each analysis task: CORAS [15], [71] for risk analysis, the security requirements engineering framework by Haley et al. [16] for the design process, and Microsoft's so-called threat modeling with data flow diagrams (DFD) and STRIDE [6], [72] for security design analysis.

A. Risk Analysis: CORAS

CORAS [15], [71] is a model-driven approach to risk analysis comprising a graphical modeling language and a method of creating and evaluating models. CORAS models represent causal chains of events with adversaries and other causes of risks at one end, assets to be protected at the other, and threat scenarios with their expected consequences on paths between them. Fig. 4 shows as an illustration some of the elements of a possible model for the example used before. These graphs can be annotated with information such as the likelihood of a threat and its respective consequences, the impact on an asset of interest, and how acceptable or unacceptable consequences are. Known or suspected vulnerabilities and treatments to reduce risks can also be represented in CORAS models, but only as abstract annotations for the purpose of assessing their impact on the overall risk model. Beyond that, system design is not modeled or considered.

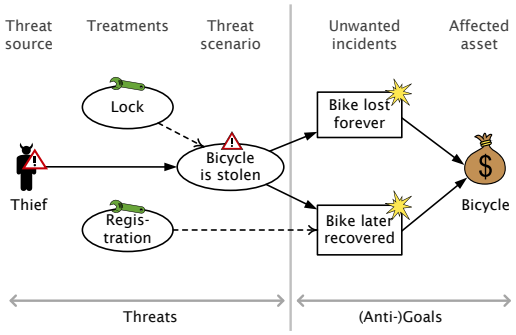

Figure 4. A CORAS diagram sketch. CORAS models represent scenarios how threats may lead to undesired consequences.

CORAS builds on risk management standards and takes an asset-driven approach. Assets to be preserved imply security goals. Human and technical sources pose deliberate and accidental threats. CORAS models capture scenarios how these threats may lead to undesired consequences for one or more assets, contrary to the security goals. One can then propagate probability and impact assessments through the models to calculate, for example, the contribution of a particular threat or the impact of particular mitigations on the total risk in a cost-benefit analysis. CORAS models relationships between abstract entities (e.g., "threats", "assets") and leaves it to its users to define and assess them. Finding vulnerabilities or mitigations is not specifically supported, but suspected or proposed ones can be evaluated.

B. Design Process: Security Requirements Engineering Framework

Haley et al. [16] propose a framework for the representation and analysis of security requirements. Their analysis process comprises four activities: (1) identify functional requirements, (2) identify security goals, (3) identify security requirements, and (4) construct satisfaction arguments. In line with the Twin Peaks model [17], this process is to be iterated until satisfaction arguments can be made for all security goals. Iterations may include adjustments to goals and requirements if security requirements are not feasible to satisfy through system design choices. Satisfaction arguments verify through semi-formal reasoning that a set of design choices fulfills the respective goals.

Security goals are to be expressed in terms of assets and properties to protect. Threats appear only as reversals of

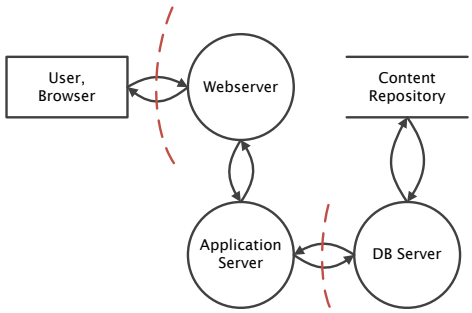

Figure 5. A data flow diagram for security design analysis. A subset of the attacks represented by the acronym STRIDE is applied to each model element.

security goals [49], because security design should aim to prevent undesired conditions altogether, rather than only overly specific attacks. Their approach thus starts, roughly, from what is represented on the right-hand side of a CORAS model, but does not consider risk analysis itself. As output one gets detailed security requirements—both functional requirements and design constraints—that are specific enough to guide system design and later verification with respect to security. An approach to security design itself is not part of the framework. However, satisfaction arguments serve to verify whichever design choices have been made during design and the framework considers also the possibility that satisfaction may be infeasible. This framework belongs at the cross-section of security goals and design as it aims to translate security goals into specific security requirements and thus into security design.

C. Security Design Analysis: DFDs and STRIDE

Microsoft developed a security design analysis technique [6], [72] based on data flow diagrams (DFDs) as part of its security development lifecycle (SDL) [73], [74]. This technique is to be applied by software development teams to identify security concerns and devise mitigations early in the development process. Microsoft is offering a free threat modeling tool, which, perhaps together with its simplicity, has popularized the technique. It has undergone extensions and updates [7], [8] and is being studied by researchers [75].

Excluding validation, security design analysis by Microsoft's original technique has three steps. First, the software or system under development is modeled in one or more data flow diagrams. Fig. 5 shows a high-level example for a generic web application; more detail can be added if needed. A DFD describes the architecture from a runtime perspective as a collection of processes, data stores, external entities, and data flows between them. In addition, trust and other boundaries can be represented by dashed lines.

The second step is rather mechanistic, assigning a selection of STRIDE attacks to each model element. The result is a list of abstract attacks, such as "spoofing of a client or user" or "tampering with the data flow from client to server." In the third step, the developers assess the impact of each item on this list and devise mitigations. The analysis technique does not prescribe how to do this, but the modeling tool offers hints and helpful questions.

Microsoft's designation of this approach as "threat modeling" is a bit of a misnomer, as the threat model is the static collection of abstract attacks encoded in STRIDE. The technique takes this conception of attacker behavior and explores a system architecture with it. Design guidance can be derived if one accepts the implied goal of preventing STRIDE attacks.

Table II
PERSPECTIVES ON SECURITY NEEDS.

Perspective	Focus
Threats	Describe adversary objectives, capabilities, and behaviors
Risk analysis	Find and prioritize conflicts between threats and security goals
Goals	Describe conditions to maintain and events to prevent
Design process	Translate goals into design decisions
Design	Describe security functions, features, and architecture
Security design analysis	Understand the actual security properties and possible attacks

VII. THE SECURITY TRIPLET PEAKS

Wrapping up, we can take six different partial perspectives on the security needs of a system (Table II). Three perspectives concern the elementary dimensions of threats, security goals, and security design, and the other three focus on pairs of dimensions and their interaction. Threats and goals together define what we may call security requirements. A system design that aims to fulfill these requirements can, however, not only fail to do so, but also change those requirements by attracting different threats and requiring further security goals.

Our discussion suggests to think of security requirements engineering in terms of a triplet peaks model, which extends the Twin Peaks model [17] with threat models as the third peak. The three analysis tasks with their respective techniques tie together three types of artifacts: threat models, security goals, and the security design of a system. Each artifact represents a distinct dimension of a system's security needs; they are interdependent, as any change in one of them affects the others. Threats determine what it takes to meet security objectives, security goals determine what counts as a threat and how important different threats are, and security design determines how threats manifest themselves and cause, or do not cause, consequences.

Consider as an example the design decision to support the Transport Layer Security (TLS) protocol [76] in a distributed system. This protocol offers confidentiality, integrity, and authentication over untrusted networks. TLS may be used for one or more of these capabilities to pursue specific security

goals and to address specific threats, or with a less specific motivation as a state-of-the-art hygiene measure. The pursued security goals may concern application assets or secondary entities that exist only as the result of previous design decisions, such as passwords. The decision to use TLS affects all three security dimensions:

- Security design—Regardless of the original motivation, all TLS capabilities become available as a basis for further design decisions, which will likely rely on it. On the other hand, the decision to use TLS entails a number of further necessary decisions, such as which cipher suites to support and how to manage certificates and private keys.
- Security goals—New security goals arise as TLS introduces new entities that need protection. For example, TLS depends on the secrecy of private keys, which must be securely generated and need appropriate protection throughout their lifecycle.
- Threats—TLS depends on a system of certification authorities (CAs) to verify identities and issue certificates. Due to this dependency, the introduction of TLS leads to new threats to consider, such as attacks against CAs. On the other hand, the network threats that TLS addresses lose weight in further security considerations.

Due to these interdependencies, security requirements cannot be fully understood in a linear, top-down process. Rather, security requirements engineering has to iterate through the dimensions and the analysis tasks.

The Twin Peaks model [17] of simultaneous, incremental development of requirements and architectures connects the idea of agile software development [77], [78] with requirements engineering. While requirements conceptually precede and guide design, requirements engineering does not prescribe waterfall development. Development rather iterates through requirements engineering and architecture until it converges toward an acceptable result. This model has been considered for security requirements, explicitly in a security adaptation of it [18] and implicitly by work on security requirements engineering referencing the model and taking iteration into account [16].

Security adds threats as a third component that independently calls for iteration. Peculiar to security are intelligent, adaptive adversaries [47]. Their interests and behaviors determine the need for and the success of security design, but they cheat [48] and adapt their attacks to the vulnerabilities of their targets. Threat models as a third peak reflect this. Security requirements engineering remains incomplete if it considers only threats and security design or only security goals and security design. The former results in overspending on defense against irrelevant threats, the latter in a false sense of security as adversary behavior is not taken into account.

VIII. Epistemic Challenges

Epistemic challenges of security requirements engineering concern our ability to anticipate security needs that arise from the intent and activity of intelligent adversaries. As a motivating example consider the CORAS diagram in Fig. 6, which describes a threat-risk model for a farm, in which cattle as an asset are threatened by alien abduction. Within this model we can reason: Propagating the probability of alien visits through the model, we conclude that our cattle are safe while grazing and there is no need to hide them. But how do we know the model is correct and complete and how do we find it and its inputs in the first place?

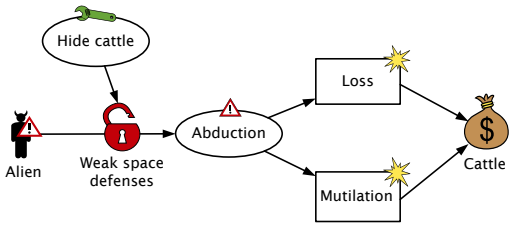

Figure 6. Should farmers use this risk model? Fears are easy to elicit, but security engineering needs sound threat models.

In general requirements engineering, requirements are validated at the latest when a system has been built that satisfies the needs of its stakeholders. Security requirements and their design consequences, however, are being tested only when a system meets its attackers. Failures or limitations then become more likely visible through detected attacks than successes preventing attacks from even being tried. This is not merely a practical challenge; there are rather fundamental questions regarding the elicitation of security needs.

A. Absent Adversaries

General requirements engineering revolves around stakeholders, who are the sources of requirements. When it comes to security requirements, however, the most important class of actors remains unavailable: Adversaries as anti-stakeholders, whose objectives, capabilities, and behaviors define the threats a system is facing, do not usually participate in efforts to figure out how to stop them. Threats can therefore be explored only indirectly.

How to nonetheless obtain reliable threat models remains an open problem. Microsoft's threat modeling technique circumvents this problem by positing STRIDE as an abstract, generic attacker model. Further interpretation and assessment is left to the user of the technique. As an advantage, the STRIDE model can be used right away; as a downside, it misses many aspects of adversary behavior and possible attacks [75].

CORAS addresses the problem of absent adversaries by ignoring it. CORAS models are to be created based on the knowledge of available stakeholders. This is not wrong per se, as domain experts may indeed know a lot about threats that matter in practice. However, the challenge remains to distinguish good from bad threat conceptions coming from these proxies. For example, empirical studies of attack trees and misuse cases, two modeling techniques belonging to the threat-design intersection, found hints (but no strong evidence)

that their participants seemed to recite what the authors called "textbook threats" rather than producing specific analyses [79], [80]. Are there better ways to capture, understand, and represent adversary behavior to support the analysis of security needs?

B. Complexity

Security has many facets. As illustrative examples, consider the hundreds of attack and vulnerability patterns collected in the Common Attack Pattern Enumeration and Classification (CAPEC) [70] and the Common Weakness Enumeration (CWE) [43], or the dozens of pages of checklists in the OWASP Application Security Verification Standard [9]. Practitioners are facing a vast number of potential concerns, of which only a subset is relevant for any system and task at hand, and complicated selection rules for these relevant items. Some attacks or vulnerabilities pertain to particular security mechanisms, some to languages or platforms, some to specific threats or environments, some to architectural patterns, and some to general hygiene. Identifying the relevant set of security concerns for a particular development situation remains an open problem, the more so as system designs themselves are complex.

Complexity affects in particular visual notations, which many security requirements techniques include. As the informal *Deutsch limit* [81] suggests, only about 50 visual primitives fit on a single screen and besides that, the cognitive limits of humans must be taken into account. These limits dictate either a rather high degree of abstraction or a narrow field of application. CORAS addresses this problem by abstraction and vagueness—the relationship between risks in a CORAS model and any design feature of a system remains unclear. Microsoft's threat modeling works with the abstract perspective of data flow diagrams, but lets complexity resurface in the checklists generated from these models. The approach of Haley et al. limits generality making assumptions about security problems and their solutions. Deeper insight into the sources of complexity in security requirements engineering and into ways of addressing complexity could be highly beneficial for practical applications.

C. Reasoning

The three analysis tasks require reasoning about facts or assumptions as their inputs to produce conclusions. Analysis techniques prescribe to a varying degree how to reason about their respective subject matter. CORAS has formal reasoning built into it, as it allows risk assessments to be propagated through its graph. Microsoft's threat modeling, on the other hand, limits formal reasoning to the mechanistic step of applying STRIDE to a data flow diagram and leaves any further thinking to its users. The approach of Haley et al. lies somewhere between these extremes, combining formal logic with less formal structured arguments.

While it is tempting to call for as much rigor as possible in security requirements techniques, two issues get in the way of formal reasoning. One is the reliability of input. If a technique lets its users capture arbitrary assumptions, as demonstrated with CORAS in Fig. 6, no amount of rigor in later reasoning can fix input errors. A practical challenge in the design of security requirements techniques is therefore how much prescriptive reasoning to build into them.

The other issue is a more fundamental one: We still do not understand the logic of security very well for real-world systems. One symptom is the notorious difficulty of reliably quantifying security [82]. As an example, Bozorgi et al. [83] analyzed the performance of a vulnerability scoring scheme, CVSS (Common Vulnerability Scoring System, https://www.first.org/cvss), as a predictor for how likely a known vulnerability would be exploited in the wild. They found CVSS to perform poorly and much worse than a machine-learning system trained with the texts of vulnerability reports. Their study concerned known vulnerabilities in deployed products; some of the information they used may not even be available at earlier stages of development.

Although rigorous formal methods seem desirable for security, they are currently unrealistic outside narrow niches. Less formal approaches, focusing on gathering and representing approximate information and leaving security expertise to the experts, may be more useful in practice than attempts to build problem solving into a requirements technique.

IX. Conclusion

Security requirements engineering inherits all the challenges of requirements engineering at large, and adds some of its own. Whether security needs have been sufficiently addressed depends not only on requirements and system design, but also on the reaction of adaptive adversaries to the design of a system. Security requirements should therefore be considered as a dynamic system with three degrees of freedom. One of them—threats—cannot be directly influenced and also evades participation in requirements engineering activities. It must nevertheless be considered by means of threat models. To succeed, security requirements engineering needs to iterate through the components of security needs and analyze their mutual influences until models converge. The complexity of security and a lack of universal rules and models for reasoning about it remain open challenges.

Acknowledgment

I thank my colleagues Philipp Holzinger, Stefan Triller, and Andreas Poller for many fruitful discussions that lead to the writing of this paper, and just as much for the fruitless ones. I also thank the reviewers for their constructive feedback on the submitted version.

References

[1] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements Eng.*, vol. 15, no. 1, pp. 7–40, 2010.

[2] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Comput. Standards & Interfaces*, vol. 32, no. 4, pp. 153–165, 2010.

[3] I. A. Tøndel, M. G. Jaatun, and P. H. Meland, "Security requirements for the rest of us: A survey," *IEEE Software*, vol. 25, no. 1, pp. 20–27, Jan. 2008.

[4] A. Souag, R. Mazo, C. Salinesi, and I. Comyn-Wattiau, "Reusable knowledge in security requirements engineering: a systematic mapping study," *Requirements Engineering*, vol. 21, no. 2, 2016.

[5] N. R. Mead, E. D. Hough, and T. R. Stehney II, "Security quality requirements engineering (SQUARE) methodology," CMU SEI, Pittsburgh, PA, Tech. Rep. CMU/SEI-2005-TR-009, 2005.

[6] F. Swiderski and W. Snyder, *Threat Modeling*. Microsoft Press, 2004.

[7] D. Dhillon, "Developer-driven threat modeling: Lessons learned in the trenches," *IEEE Security Privacy*, vol. 9, no. 4, pp. 41–47, 2011.

[8] A. Shostack, *Threat Modeling: Designing for Security*. Wiley, 2014.

[9] *Application Security Verification Standard*, OWASP, 2016, v3.0.1. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project

[10] J. Whitmore, S. Türpe, S. Triller, A. Poller, and C. Carlson, "Threat analysis in the software development lifecycle," *IBM J. Research Develop.*, vol. 58, no. 1, pp. 6:1–6:13, 2014.

[11] L. Chung, "Dealing with security requirements during the development of information systems," in *Proc. Adv. Inform. Syst. Eng. (CAiSE'93)*. Springer, 1993, pp. 234–251.

[12] D. G. Firesmith, "Engineering security requirements," *J. Object Technol.*, vol. 2, no. 1, 2003. [Online]. Available: http://www.jot.fm/issues/issue_2003_01/column6.pdf

[13] D. Ameller, C. Ayala, J. Cabot, and X. Franch, "Non-functional requirements in architectural decision making," *IEEE Softw.*, vol. 30, no. 2, pp. 61–67, 2013.

[14] D. Firesmith, "Specifying reusable security requirements," *J. Object Technol.*, vol. 3, no. 1, pp. 61–75, 2004. [Online]. Available: http://www.jot.fm/issues/issue_2004_01/column6.pdf

[15] M. S. Lund, B. Solhaug, and K. Stølen, *Model-Driven Risk Analysis - The CORAS Approach*. Springer, 2011.

[16] C. B. Haley, R. Laney, J. Moffett, and B. Nuseibeh, "Security requirements engineering: A framework for representation and analysis," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 133–153, 2008.

[17] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–119, 2001.

[18] T. Heyman, K. Yskout, R. Scandariato, H. Schmidt, and Y. Yu, "The security twin peaks," in *Engineering Secure Software and Systems*, Ú. Erlingsson, R. Wieringa, and N. Zannone, Eds. Springer Berlin / Heidelberg, 2011, pp. 167–180.

[19] R. J. Anderson, *Security Engineering: A guide to building dependable distributed systems*. Wiley, 2008.

[20] B. W. Lampson, "Computer security in the real world," *Computer*, vol. 37, no. 6, pp. 37–46, 2004.

[21] ——, "Protection," in *Proc. 5th Princeton Symp. on Inform. Sci. and Syst.* ACM, 1971, pp. 437–443.

[22] J. H. Saltzer, "Protection and the control of information sharing in multics," *Commun. ACM*, vol. 17, no. 7, pp. 388–402, 1974.

[23] S. R. Ames Jr., M. Gasser, and R. R. Schell, "Security kernel design and implementation: An introduction," *Computer*, vol. 16, no. 7, pp. 14–22, 1983.

[24] D. D. Clark and D. R. Wilson, "A comparison of commercial and military computer security policies," in *IEEE Symp. Security and Privacy*, Apr. 1987.

[25] S. Lipner, T. Jaeger, and M. E. Zurko, "Lessons from vax/svs for high-assurance vm systems," *IEEE Security Privacy*, vol. 10, no. 6, pp. 26–35, 2012.

[26] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975.

[27] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[28] R. Merkle, "Secure communications over insecure channels," *Commun. ACM*, vol. 21, no. 4, pp. 294–299, Apr. 1978.

[29] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[30] R. E. Smith, "A contemporary look at Saltzer and Schroeder's 1975 design principles," *IEEE Security Privacy*, vol. 10, no. 6, pp. 20–25, 2012.

[31] M. Bishop, S. Engle, D. Howard, and S. Whalen, "A taxonomy of buffer overflow characteristics," *IEEE Trans. Depend. Sec. Comput.*, vol. 9, no. 3, pp. 305–317, 2012.

[32] L. Szekeres, M. Payer, T. Wei, and D. Song, "SoK: Eternal war in memory," in *Proc. 2013 IEEE Symp. Security and Privacy*, 2013, pp. 48–62.

[33] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[34] J. E. Forrester and B. P. Miller, "An empirical study of the robustness of Windows NT applications using random testing," in *Proc. 4th USENIX Windows Syst. Symp.*, 2000, pp. 59–68.

[35] R. Anderson, "Why cryptosystems fail," in *Proc. 1st ACM Conf. on Comput. and Commun. Security (CCS'93)*. New York, NY, USA: ACM, 1993, pp. 215–227.

[36] S. Garfinkel, G. Spafford, and A. Schwartz, *Practical UNIX and Internet Security*, 3rd ed. O'Reilly, 2003.

[37] K. M. Goertzel, T. Winograd, H. L. McKinley, L. J. Oh, M. Colon, T. McGibbon, E. Fedchak, and R. Vienneau, "Software Security Assurance: A State-of-the-Art Report," IATAC & DACS, Tech. Rep., 2007. [Online]. Available: http://www.dtic.mil/docs/citations/ADA472363

[38] B. Blakley and C. Heath, "Security design patterns," The Open Group, Technical Guide G031, Apr. 2004.

[39] M. Schumacher, *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2006.

[40] K. Yskout, R. Scandariato, and W. Joosen, "Do security patterns really help designers?" in *Proc. 37th Int. Conf. on Softw. Eng. (ICSE'15)*, vol. 1. IEEE, 2015, pp. 292–302.

[41] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 371–386, 2011.

[42] M. Stytz, "Considering defense in depth for software applications," *IEEE Security Privacy*, vol. 2, no. 1, pp. 72–75, Jan. 2004.

[43] "Common weakness enumeration (CWE)," version 2.11. [Online]. Available: http://cwe.mitre.org/

[44] Z. Durumeric, M. Payer, V. Paxson, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, and J. Beekman, "The matter of Heartbleed," in *Proc. 2014 Internet Measurement Conf. (IMC'14)*. New York, New York, USA: ACM Press, 2014, pp. 475–488.

[45] L. Cranor and S. Garfinkel, Eds., *Security and Usability*. O'Reilly, 2005.

[46] S. Garfinkel and H. R. Lipford, "Usable security: History, themes, and challenges," *Synthesis Lectures on Information Security, Privacy, and Trust*, vol. 5, no. 2, pp. 1–124, 2014.

[47] C. Severance, "Bruce Schneier: the security mindset," *Computer*, vol. 49, no. 2, pp. 7–8, Feb. 2016.

[48] G. Conti and J. Caroland, "Embracing the Kobayashi Maru: why you should teach your students to cheat," *IEEE Security Privacy*, vol. 9, no. 4, pp. 48–51, 2011.

[49] C. B. Haley, R. C. Laney, and B. Nuseibeh, "Deriving security requirements from crosscutting threat descriptions," in *Proc. 3rd Int. Conf. on Aspect-oriented Softw. Develop. (AOSD'04)*. New York, NY, USA: ACM, 2004, pp. 112–121.

[50] H. H. Thompson, "Why security testing is hard," *IEEE Security Privacy*, vol. 1, no. 4, pp. 83–86, 2003.

[51] "ISO/IEC/IEEE International Standard – systems and software engineering – life cycle processes – requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, pp. 1–94, 2011.

[52] R. Crook, D. Ince, Luncheng Lin, and B. Nuseibeh, "Security requirements engineering: when anti-requirements hit the fan," in *Proc. IEEE Joint Int. Conf. on Requirements Eng.* IEEE Comput. Soc, 2002, pp. 203–205.

[53] A. van Lamsweerde, S. Brohez, R. De Landtsheer, and D. Janssens, "From system goals to intruder anti-goals: Attack generation and resolution for security requirements engineering," in *Proc. RE'03 Workshop on Requirements for High Assurance Systems (RHAS'03)*, 2003.

[54] R. Mohanani, P. Ralph, and B. Shreeve, "Requirements fixation," in *Proc. 36th Int. Conf. on Softw. Eng. (ICSE'14)*. New York, NY, USA: ACM, 2014, pp. 895–906.

[55] "Internet security glossary, version 2," IETF, RFC 4949, Aug. 2007. [Online]. Available: http://tools.ietf.org/html/rfc4949

[56] A. Poller, S. Türpe, and K. Kinder-Kurlanda, "An asset to security modeling? analyzing stakeholder collaborations instead of threats to assets," in *Proc. 2014 New Security Paradigms Workshop (NSPW'14)*, 2014, pp. 69–82.

[57] D. Van Lierop, M. Grimsrud, and A. El-Geneidy, "Breaking into bicycle theft: Insights from montreal, canada," *Int. J. Sustainable Transportation*, vol. 9, no. 7, pp. 490–501, Oct. 2015.

[58] C. E. Landwehr, "Computer security," *Int. J. Inform. Security*, vol. 1, no. 1, pp. 3–13, 2001.

[59] H. H. Willis, A. R. Morral, T. K. Kelly, and J. J. Medby, "Estimating terrorism risk," Center for Terrorism Risk Management Policy, Rand Corp., Tech. Rep., 2005. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a449118.pdf

[60] "Common criteria for information technology security evaluation," Sep. 2012. [Online]. Available: http://www.commoncriteriaportal.org/cc/

[61] D. Firesmith, "Common concepts underlying safety, security, and survivability engineering," Carnegie Mellon Univ., Technical Report CMU/SEI-2003-TN-033, 2003.

[62] S. Fenz and A. Ekelhart, "Formalizing information security knowledge," in *Proc. 4th Int. Symp. on Inform., Comput., and Commun. Security (ASIACCS'09)*. New York, NY, USA: ACM, 2009, pp. 183–194.

[63] É. Dubois, P. Heymans, N. Mayer, and R. Matulevičius, "A systematic approach to define the domain of information system security risk management," in *Intentional Perspectives on Information Systems Engineering*, S. Nurcan, C. Salinesi, C. Souveyet, and J. Ralyté, Eds. Springer, 2010, pp. 289–306.

[64] C. Herley and W. Pieters, "'If you were attacked, you'd be sorry': Counterfactuals as security arguments," in *Proc. 2015 New Security Paradigms Workshop (NSPW'15)*. New York, New York, USA: ACM Press, 2015, pp. 112–123.

[65] R. Sonnenschein, A. Loske, and P. Buxmann, "Which IT security investments will pay off for suppliers? using the Kano model to determine customers' willingness to pay," in *49th Hawaii Int. Conf. on Syst. Sci. (HICSS'16)*, Jan. 2016, pp. 5672–5681.

[66] S. Berkun, *Making Things Happen: Mastering Project Management*. O'Reilly, 2008.

[67] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security Privacy*, vol. 3, no. 1, pp. 84–87, 2005.

[68] D. Geer and J. Harthorne, "Penetration testing: a duet," in *Proc. 18th Annu. Comput. Security Appl. Conf. (ACSAC'02)*, 2002, pp. 185–195.

[69] C. C. Palmer, "Ethical hacking," *IBM Syst. J.*, vol. 40, no. 3, pp. 769–780, 2001.

[70] "Common attack pattern enumeration and classification (CAPEC)," version 2.10. [Online]. Available: http://capec.mitre.org/

[71] B. Solhaug and K. Stølen, "The CORAS language–why it is designed the way it is," in *Proc. 11th Int. Conf. on Structural Safety and Reliability (ICOSSAR'13)*. Taylor & Francis, 2013, pp. 3155–3162.

[72] A. Shostack, "Experiences threat modeling at Microsoft," in *Modeling Security Workshop*. Dept. of Computing, Lancaster University, UK, 2008.

[73] S. Lipner, "The trustworthy computing security development lifecycle," in *Proc. 20th Annu. Comput. Security Appl. Conf. (ACSAC'04)*, 2004, pp. 2–13.

[74] M. Howard and S. Lipner, *The Security Development Lifecycle*. Microsoft Press, 2006.

[75] R. Scandariato, K. Wuyts, and W. Joosen, "A descriptive study of Microsoft's threat modeling technique," *Requirements Eng.*, vol. 20, no. 2, pp. 163–180, June 2015.

[76] S. Turner, "Transport layer security," *IEEE Internet Computing*, vol. 18, no. 6, pp. 60–63, Nov. 2014.

[77] J. Highsmith and A. Cockburn, "Agile software development: the business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, Sep. 2001.

[78] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001. [Online]. Available: http://agilemanifesto.org/

[79] A. L. Opdahl and G. Sindre, "Experimental comparison of attack trees and misuse cases for security threat identification," *Inform. and Softw. Technol.*, vol. 51, no. 5, pp. 916–932, 2009.

[80] P. Karpati, Y. Redda, A. L. Opdahl, and G. Sindre, "Comparing attack trees and misuse cases in an industrial setting," *Inform. and Softw. Technol.*, vol. 56, no. 3, pp. 294–308, 2014.

[81] "comp.lang.visual FAQ," 1998. [Online]. Available: http://www.faqs.org/faqs/visual-lang/faq/

[82] V. Verendel, "Quantified security is a weak hypothesis: a critical survey of results and assumptions," in *Proc. 2009 New Security Paradigms Workshop*. New York, NY, USA: ACM, 2009, pp. 37–50.

[83] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proc. 16th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'10)*. New York, NY, USA: ACM, 2010, pp. 105–114.