Journal of Information Security and Applications 89 (2025) 103954

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

BridgeSec: Facilitating effective communication between security engineering and systems engineering

Avi Shaked [a],*, Nan Messe [b]

[a] Department of Computer Science, University of Oxford, Oxford, UK
[b] IRIT, Toulouse University, CNRS, INP, UT2, Toulouse, France

ARTICLE INFO

Keywords:
Security by design
Model-driven engineering
System development
Vulnerability management
Threat modelling
Security engineering
Systems engineering

ABSTRACT

We increasingly rely on systems to perform reliably and securely. Therefore, it is imperative that security aspects are properly considered when designing and maintaining systems. However, achieving the security by design ideal is challenging. Security information is typically unstructured, dispersed, hard to communicate, and its assessment is somewhat subjective and tacit. Additionally, the inclusion of security information within design requires integrating the efforts of two knowledge-intensive disciplines: security engineering and systems engineering. In this paper, we introduce BridgeSec, a novel conceptual information-exchange interface to systemise the communication of security information between these two disciplines. The main contribution of BridgeSec lies in its explicit identification of concepts related to vulnerability management, which allows systems engineering and security engineering teams to codify pertinent information. The disciplines involved in the system design can thus coordinate policies, implementations and, ultimately, the security posture. Furthermore, based on the newly unveiled interface, an automated reasoning mechanism is specified. This mechanism allows to reason about the vulnerability posture of systems in a scalable and systematic way. First, we describe and formalise the information-exchange interface BridgeSecand how it can be used to reason about the security of systems designs. Next, we present an open-source prototype – integrated into a threat modelling tool – which rigorously implements the interface and the reasoning mechanism. Finally, we detail two diverse and prominent applications of the interface for communicating security aspects of systems designs. These applications show how BridgeSec can rigorously support the design of systems' security in two representative scenarios: in coordinating security features and policy during design, and in coordinating mitigation to disclosed implementation vulnerabilities.

1. Introduction

The pervasive nature of software-intensive systems in modern society has transformed various industries, yet the security of such systems remains a pressing concern [1]. Unfortunately, systems development efforts often prioritise functionality over security, leading to a lack of inherent security orientation [2,3]. This is particularly true in time-sensitive scenarios where small-scale companies or those aiming to be market pioneers prioritise time-to-market over security considerations [4]. Incorporating security aspects and considerations into system development is of paramount importance to ensure the resilience and integrity of the resulting systems [5]. Failing to address security concerns adequately can have severe consequences, including data breaches, service disruptions, and compromised user trust [6]. Therefore, it is essential to incorporate security as a fundamental consideration throughout the entire system development lifecycle [5,7–9].

When security is treated as an afterthought or added retroactively, it can lead to significant challenges and increased costs [10]. Previous research [11,12,9] and security policymakers' publications [7,13] emphasise the significance of integrating security into the system from its inception, rather than attempting to retrofit it later. Such integration requires that system developers participate in the decision-making processes of designing for security as well as that these developers tap into cyber security domain knowledge.

By proactively identifying and addressing security during the early phases of the system development lifecycle, particularly in the requirement and design stages, vulnerabilities can be communicated with the development team earlier. This could reduce the likelihood of cyber incidents and minimise their impacts – including operational, financial and reputational – on organisations [14]. The identified vulnerabilities can be in the form of a conceptual design weakness or a concrete

* Corresponding author.
E-mail addresses: avi.shaked@cs.ox.ac.uk (A. Shaked), nan.messe@irit.fr (N. Messe).

https://doi.org/10.1016/j.jisa.2024.103954

2214-2126/© 2024 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

implementation, as further explained in the Background section (Section 2). By adopting this security-centric approach early in the system development process, also referred to as "security by design" [15,13], organisations can build systems that are more resilient, trustworthy, and better aligned with the needs of their stakeholders. As a result, this proactive stance towards security not only enhances the overall system's trustworthiness, but also saves costs associated with addressing and fixing security issues later in the development cycle [9,16].

To address security concerns from the outset and establish solid foundations for subsequent development efforts and operations, system architects and engineers should take proactive steps. They can specify required security functionality and requirements as well as identify appropriate security controls and embed them into their designs [5,17,3]. In practice, however, system engineers often lack sufficient expertise in cyber security [18]. Thus, performing security by design often necessitates the involvement of dedicated security specialists within the development team [19].

Consequently, achieving the ideal of security by design is challenging due to the need to balance two perspectives: the security engineering view and the systems engineering view [20,21,17,22–24]. The two disciplines often use different terminologies and relate to their domain-specific concepts. For example, system engineers typically communicate design aspects by expressing the system's functionalities, subsystems, components and connectors [25,26], using modelling languages such as SysML [27]. In contrast, security experts communicate the security posture using concepts such as vulnerabilities, attacks and mitigation/controls [28–30], represented by artefacts such as attack trees [31] and threat models [32–34]. This disconnection can result in security considerations being overlooked or inadequately communicated and consequently poorly addressed, leaving the system vulnerable to potential threats.

Furthermore, system designers might exercise inept judgment due to poor security orientation, insufficient training or conflict with explicit key performance indicators measurements (or rather, non-existent key performance indicators with respect to security [3]). As security compliance of individuals remains a major challenge, organisations need to adopt appropriate management practices that support security-related individual decision-making [35].

Moreover, a recent systematic mapping study on security for system of systems establishes the lack of strategic, widely applicable approaches to planning and coordinating the security posture of a system and its constituents [36]. Specifically, the study suggests that future work should address technical management aspects relating to: (a) the responsibility of analysing the security of the systems; (b) the responsibility and approaches to develop countermeasures to prevent exploitation of detected vulnerabilities; (c) how to coordinate the security of systems in a dynamic environment.

To address the serious gaps in coordinating security in systems development contexts, it is crucial to facilitate effective, ongoing communication and collaboration between system engineers and security experts. This can be achieved by establishing a shared language or a set of concepts that can be understood by both parties [37,38]. A common understanding allows for the seamless exchange of information, such as ideas, requirements, and concerns. System development is knowledge-intensive, and it is essential to codify pertinent knowledge to support effective collaboration between stakeholders [39,40], including system engineers and security experts. An interface for exchanging information – henceforth, an "information-exchange interface" – is therefore an important design aspect of organisational operation, which can facilitate tapping into internal and external knowledge and team-reasoning [41, 40].

Moreover, security posture and compliance are affected by how stakeholders (such as system engineers) perceive security policies, and it is, therefore, important to communicate security policies and expectations in a clear, well-formed way using a clear-cut process [42]. Using an information exchange interface between systems engineering and security experts, the development process can benefit from clear articulation of security policies and objectives, enhanced coordination, improved decision-making, and more efficient integration of security measures into systems [37].

In this article, we address the challenges outlined above by providing a mechanism to consolidate the systems engineering and the security engineering views, and to systematically support security by design. We propose the use of an explicit information-exchange interface – BridgeSec – between the two disciplines as well as provide formal definitions of related concepts and a semi-automated reasoning engine. BridgeSec is specifically designed to assist organisations in creating secure systems, by: (1) employing a divide-and-conquer strategy for the design effort, shared between systems engineering and security teams; (2) providing and communicating well-structured security policies and system designs between the two teams. These ensure a disciplined approach to security by design, as well as rigorous decision-making with respect to the security posture of the systems that are being developed or deployed.

From the security engineering perspective, BridgeSec supports the creation of security policies or specifications of security requirements to be included in system development efforts. Such definitions can codify expert knowledge. These definitions can be aligned with the types of components integrated into the design, as specified by systems engineers. Security experts can suggest necessary security controls – to be considered as security requirements – during the design of systems, ensuring alignment with security objectives.

From the system engineering perspective, BridgeSec allows to explicitly communicate the identified security controls (those needed to be integrated into the system design to improve the security posture of systems and/or meet the security policy's requirements, as specified by security experts), with respect to the actual system design. BridgeSec can underpin real-time feedback and recommendations with respect to the system's security posture, empowering system engineers to make informed decisions that enhance the security of their systems.

To demonstrate the feasibility and practicality of the suggested conceptual, information-exchange interface, we provide a rigorous prototype of BridgeSec, using model-based technologies that employ meta-modelling and model-based representations. We illustrate the application of BridgeSec, using our prototype, in two widely applicable scenarios. These examples showcase BridgeSec's value and ability to accommodate the design in different levels of abstraction and in various stages of the system life cycle, as well as to make informed, security-related decisions about the design.

The contribution of this research is a novel information-exchange interface – BridgeSec – to support security by design. More precisely, this contribution is threefold: (1) identifying an information-exchange interface that allows communicating security aspects between the security engineering and the systems engineering disciplines as well as supporting the separation of concerns between the disciplines. The interface comprises two essential concepts: component types and security controls; (2) articulating and formalising a reasoning mechanism, which utilises the identified interface as a means to achieve security by design. Specifically, the interface and reasoning mechanism are designed to support a vulnerability management approach, which is essential to security by design. [43,19]. Also, they can incorporate information from security knowledge bases and system models, as well as support their integration.; (3) implementing a well-defined, open-source prototype of BridgeSec. The prototype extends the state-of-the-art security by design methodology and tool – TRADES [17, 44]. Specifically, we introduce new concepts related to vulnerability management into TRADES, integrate them with existing concepts, and provide a graphical representation that employs the interface and reasoning mechanism.

This paper is structured as follows: in Section 2, we provide pertinent background about the existing security body of knowledge and security by design. As this work revises TRADES [17] and significantly

extends it, we also provide pertinent TRADES background in this section. Next, BridgeSec is described and formalised in Section 3. Then, in Section 4, we discuss realistic, representative examples of applying BridgeSec. We conclude by situating our work with respect to the state-of-the-art in Section 5, and finally, in Section 6, by discussing the proposed information-exchange interface and its applications as well as potential extensions and future work.

2. Background

2.1. Security body of knowledge

Several common security knowledge bases are available as information sources to understand and manage vulnerabilities and risks. These include CWE, CVE, CPE, NIST SP 800–53 , CAPEC and ATT&CK. We shortly describe them.

CWE (Common Weakness Enumeration) [45] is a comprehensive list of common software and hardware weaknesses. It provides a standardised language for describing and categorising vulnerabilities, enabling security professionals to identify, understand, and address them effectively. Each CWE record represents a conceptual vulnerability. This helps create awareness about common security pitfalls and guides the development of secure software and systems.

CVE (Common Vulnerabilities and Exposures) [46] is a dictionary-like database of publicly-known information security vulnerabilities. Each CVE record represents a concrete, implementation-specific vulnerability. A CVE record is typically the manifestation of one or more conceptual vulnerabilities identified in CWE. Each CVE vulnerability is assigned a unique identifier, allowing for easy reference and tracking across different systems and products. CVE provides a common platform to understand, communicate and share vulnerability information, assisting organisations in prioritising and mitigating potential risks.

CPE (Common Platform Enumeration) [47] is a structured naming scheme that facilitates the identification and description of software applications and hardware devices. It enables the standardised representation of vendor, product, and version information, aiding in the accurate identification and tracking of components across different systems. CPE supports vulnerability management and assessment by providing a consistent and uniform way to catalogue software and hardware assets. CVE records typically refer to impacted assets by using their CPE identification.

NIST SP 800-53 (National Institute of Standards and Technology Special Publication 800-53) [48] is a comprehensive publication that provides a set of security controls and guidelines for information systems and organisations. It offers a framework for managing security risks and protecting sensitive information. NIST SP 800-53 covers various security domains, including access control, incident response, cryptography, secure development, and others. It serves as a valuable resource for organisations in designing, implementing, and assessing security controls to safeguard systems.

CAPEC (Common Attack Pattern Enumeration and Classification) [49] and ATT&CK (Adversarial Tactics, Techniques & Common Knowledge) [50] are two knowledge bases dedicated to classifying and describing attacks. Attacks typically exploit vulnerabilities, and, accordingly, CAPEC attack patterns are linked with the CWE records that they potentially exploit. MITRE – the nonprofit corporation that develops and maintains CAPEC and ATT&CK – explicitly identifies that ATT&CK is a more computer-network-oriented body of knowledge and that it includes techniques for employing attack patterns identified in the more general CAPEC knowledge base. An interesting aspect of ATT&CK is that – in addition to the adversarial viewpoint – it has evolved to include a more explicit identification of mitigations and assets. ATT&CK mitigations are *"security concepts and classes of technologies that can be used to prevent a technique or sub-technique from being successfully executed"* [50]. ATT&CK assets are *"devices and systems commonly found within Industrial Control System environments"*, with those devices including *"considerations for hardware, software, architecture, and intended function"* [50]. Mitigations and assets are mapped – in ATT&CK – to the adversarial techniques.

Fig. 1 shows the prominent information security resources and how they relate to each other. These resources collectively contribute to the security body of knowledge by providing standardised frameworks, classifications, and guidance for identifying, addressing, and managing vulnerabilities. They assist security professionals, system developers, system administrators and organisations in understanding and applying best practices to enhance the security posture of software, hardware and information systems.

While these resources individually contribute to the security body of knowledge, they do not form a cohesive, consistent body of knowledge. Each resource addresses different aspects of security and can be utilised independently based on specific needs and requirements. Furthermore, the resources are only semi-structured, and this hinders the ability to formalise them. Specifically, the MITRE knowledge bases – CWE, CVE, CAPEC and ATT&CK – do not refer explicitly to NIST SP 800-53 security controls. Instead, they offer possible mitigation for vulnerabilities or attacks using unstructured natural language, with the exception of ATT&CK which also maps to its own collection of mitigations. These resources do not provide collective systematic guidance for designing secure systems. Furthermore, even the recent guidelines by world-leading security agencies – that acknowledge the need to address vulnerabilities by design – fail to provide guidance for the systematic management of vulnerabilities and the vulnerability posture of systems [13].

2.2. Security by design culture, activities and methodologies

Two distinct strategies have been identified for integrating security into the software and system development process [51,52]. The first is an "a priori" approach, which entails incorporating security considerations throughout the System Development Life Cycle (SDLC). The second is an "a posteriori" approach, which involves addressing security aspects as an afterthought to the system development, usually by reactively applying patches [53] or by adding dedicated security products into the operational environment in which systems are deployed [54], e.g., by integrating firewall components into the network in which systems operate. The latter can be characterised as a separate effort from the system development effort, while the former is an effort inherent to systems development.

As emphasised in Section 1, it is crucial to prioritise security as a proactive measure throughout the SDLC, to mitigate potential performance degradation, reduce costs, meet time-to-market constraints, and enhance overall usability [43]. Despite the prevailing tendency to address security as an afterthought in practice [55], recent years have witnessed an increasing emphasis on adopting the "a priori" strategy to ensure that security is ingrained within the development process [15].

The concept of "security by design" [13,56] falls in the scope of the "a priori" strategy, emphasising the proactive inclusion of security measures to address security threats. It advocates for a holistic and anticipatory approach, promoting a secure by default paradigm in system component configuration, access policies, and system security assurance processes. A primary objective of the security by design paradigm is to identify potential security vulnerabilities at the earliest stages of the SDLC and to incorporate security requirements and features from the outset [57–59].

Security vulnerabilities can be conceptual design weaknesses emerging explicitly or implicitly from design decisions (e.g., CWE weaknesses). Also, even in the early stages of system design, concrete vulnerabilities can be identified, such as those that emerge from decisions to incorporate specific software products with disclosed CVE records (e.g., a specific operating system) into the system that is under development. Identifying and communicating both conceptual and concrete vulnerabilities during system design is crucial to facilitating

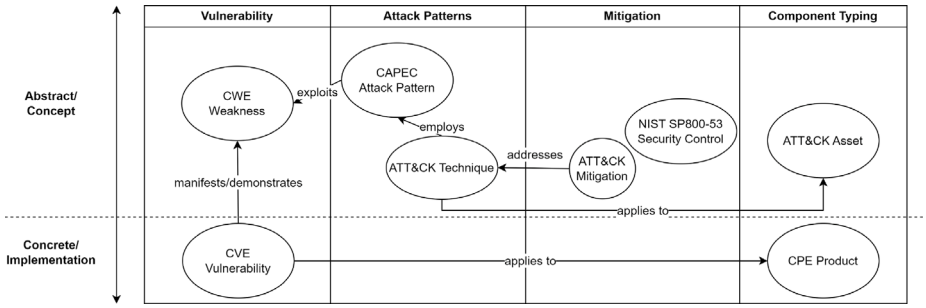

Fig. 1. Security knowledge bases and their relations.

informed decision-making about the design and the system's security posture.

The security by design approach necessitates the integration of security during the design phase through the utilisation of security assurance processes [9,57,58]. Security assurance processes involve activities such as comprehensive threat modelling, where threats are identified and communicated with the developers as well as countered through informing and educating the developers about security best practices and security controls [9]; and complementary vulnerability management, which deals with the identification, analysis, mitigation, and management of vulnerabilities in different stages of the system development [57]. Threats are the potential exploitation of vulnerabilities that exist in systems. Vulnerabilities can, therefore, be seen as the root cause enablers of threats (alternatively, attacks). Vulnerabilities are manifestations of issues in the system design. In our effort to bridge security engineering and systems engineering, we focus on vulnerability management, as the elementary security by design practice that is relevant to both disciplines.

Vulnerability management is an essential practice for developing and maintaining secure systems. It involves identifying, assessing, and mitigating vulnerabilities from the early stages of the SDLC, to build secure and resilient systems and in support of their proper maintenance in case new vulnerabilities are disclosed throughout the system life cycle. Proactive vulnerability management is typically carried out by security experts, together with system engineers, during brainstorming sessions, to identify potential vulnerabilities specific to the system's architecture, components, and functionality [19].

During the design phase, vulnerability management focuses on identifying potential attack vectors and the associated vulnerabilities that may be present in the system. This includes: (1) analysing the system's attack surface, by clearly identifying the system's constituents, such as software, hardware, and configuration components, or related processes; (2) evaluating potential weaknesses of the system, by identifying, assessing, and prioritising vulnerabilities based on their severity, potential impact, and exploitability using frameworks (such as the Common Vulnerability Scoring System [60]), and (3) defining appropriate security controls to mitigate the undesired vulnerabilities, such as employing encryption and authentication mechanisms and establishing proper access controls.

Due to the inherent variability of software-intensive systems, it is imperative to explicitly specify the security requirements that must be addressed during the design phase [59,61]. Since both the threat landscape and the system design typically evolve throughout the development life cycle, a clear specification of security requirements facilitates the creation of tailored and effective security solutions [62–64].

In general, security by design is approached through three primary methodologies: attack-tree-based, ontology-based, and model-driven security methodologies. Attack-tree-based methodologies are popularly used in security by design to analyse and assess the potential attack scenarios and vulnerabilities of a system [65,66]. The generation of an attack tree involves constructing a hierarchical tree-like structure [31,67], where the root represents the main objective, such as compromising the system's confidentiality, integrity, or availability. The tree branches out into different attack paths, representing various attack techniques or steps required to achieve the objective. Each node in the tree represents a specific attack goal or sub-goal, and the edges depict the relationships between the attack steps.

The attack-defence tree [68,69] is an extension of the attack tree that incorporates the notion of defence strategies and countermeasures into the analysis of systems security. It provides a framework for modelling and evaluating both the potential attack paths and the corresponding defence mechanisms. Similar to attack trees, attack-defence trees are structured hierarchically, with the root representing the main objective compromising the system's security. Attack-defence tree branches out into different attack paths, depicting the potential attack techniques or steps. However, unlike attack trees, attack-defence trees also include defence nodes, representing the various defence strategies and countermeasures that can be employed to mitigate the corresponding attack.

By employing attack tree-based methodologies, security professionals can systematically evaluate the security posture of a system, and identify potential weaknesses or vulnerabilities and defence strategies. These informal methodologies provide visual representations of the attack and defence paths and allow for the quantification of risks associated with each attack path. This enables security experts to prioritise their efforts and focus on mitigating the most critical threats.

Integrating the results of vulnerability analysis performed using an attack tree-based methodology into the system development process is often an informal, error-prone process. The interpretation and integration of attack tree-based results into the system development lifecycle often rely on the subjective expertise and judgment of security experts and system engineers involved. Attack tree-based methodologies offer no engineering design perspective, and by that, they exemplify the gap in the transition from security analysis to system design. Ultimately, while the insights provided by the attack tree analysis can be valuable, the translation of these findings into concrete actions and system design decisions lacks a formalised, systematic approach [52].

Formal-ontology-based methodologies can be employed in the context of security by design to enhance the understanding of security-related concepts, relationships, and properties within a system. They leverage the power of formal ontologies, which are formal models that capture domain knowledge and enable reasoning, to support the systematic integration of security considerations into the system development process [70,71]. In such methodologies, security experts utilise ontologies to define a conceptual framework that encompasses relevant security domains, such as threats, vulnerabilities, assets, and security controls, providing a structured representation of security knowledge, which, in turn, enables formal reasoning and inference capabilities [33]. Furthermore, the use of ontologies in security by design promotes knowledge reuse and information sharing across different system development projects. By capturing security knowledge in a standardised and reusable format, ontologies can facilitate the

transfer of best practices, lessons learned, and domain-specific security expertise.

Ontology-based methodologies for security by design can offer valuable insights and benefits in terms of capturing and representing security knowledge within a system. Ontologies can provide for rigorous, explicit definitions as well as for sustainable vulnerability management [61]. However, ontology-based methodologies require specialised knowledge and skills in ontology engineering and semantic reasoning, which are not widely available in systems engineering and security engineering circles.

Integrating the results of an ontology-based analysis into the system development process requires a high level of expertise in understanding and utilising ontologies effectively. Furthermore, ontology-based approaches commonly lack appropriate graphical representations to communicate systems designs and security analysis in a way that is natural to systems engineers and to security experts, both individually for each discipline and collectively. As a result, ontology-based methodologies by themselves are a limited tool for integrating security information into system development efforts.

Model-driven security is an inclusive term for specifying security aspects using well-structured models. It can be used to formalise aspects of the previously mentioned approaches (i.e., attack tree-, attack-defence tree-, and formal-ontology-based methodologies) [72]. The origin of model-driven security can be traced back to the broader concept of model-driven development [73], which aims to improve the productivity, quality, and maintainability in software and systems development, by leveraging models as primary artefacts throughout the SDLC [74]. Models can be perceived as structured *"living documents maintained to reflect design choices and system revisions"* [28]. Instead of focusing solely on writing code or creating free-form diagrams, model-driven development emphasises the creation and manipulation of structured information models that capture various aspects of the system which is being developed.

Similarly, model-driven security refers to a promising approach that leverages models as central artefacts in the design, analysis, and implementation of secure systems [75,76]. By utilising models as a shared source of information on security requirements, policies, and mechanisms, model-driven security may enable security experts to express their expertise in a structured and formal manner. These models serve as a means to capture and communicate security-related knowledge, potentially allowing security experts to articulate their security concerns, design decisions, and recommendations in a more rigorous and communicable way. System engineers can leverage these security models to gain insights into the security requirements and constraints of the system.

An important aspect of model-driven solutions is model-based representations, i.e. representations that are generated according to the underlying information model. Such representations can communicate the models in ways that are naturally or easily understood by the designated audience, e.g., by attack trees or by block diagrams for security analysts and systems engineers respectively. The representations can also convey design analysis and tips that may emerge from the use of an ontology-based approach [77,78].

Various model-driven security approaches for the design and analysis of secure software and systems exist. These include Unified Modelling Language (UML)-based approaches (e.g. SecureUML [79]), System Modelling Language (SysML)-based approaches (e.g. SysMLSec [80], MBSAES [81], and SoSSec [82]), and other Domain-Specific Languages (DSL)-based approaches (e.g. VERDICT [72], MAL [34] and the model-driven vulnerability specification approach by Rouland et al. [83]). These approaches necessitate a high level of proficiency in software and system modelling for designers to effectively conduct secure system development [17]. However, in widespread scenarios where security experts are not software or system engineers, expecting them to possess such modelling expertise becomes unrealistic.

TRADES is a model-driven methodology to address the design and analysis of systems specifically from a security perspective [17,84]. It supports a disciplined yet practical model-driven approach, including diverse representations grounded in a single well-defined metamodel to promote a holistic view. While TRADES supports the association of threats for a system of interest and its constituents as well as their risk management, it does not support vulnerability management [85]. Specifically, the representation of vulnerabilities at any level of abstraction is not supported. Also, TRADES lacks a type system for components at any level of detail (e.g., software, specific software package, and a specific version of a software package).

In Section 3.3, we present an implementation of the proposed conceptual information-exchange interface, which extends TRADES and addresses its current limitations in representing vulnerabilities and component types. We do this by introducing and integrating vulnerability management concepts into the systems security models and their representations. This implementation exhibits a model-driven security approach that incorporates an ontology-based approach, accommodates information from existing knowledge bases (e.g., CWE, CVE, and CPE), and communicates the results of the ontology-based reasoning with systems engineers and security experts on the fly – during the design process as it evolves – without requiring them to have any knowledge of formal ontologies.

3. BridgeSec: an information-exchange interface for security by design

To advance the security by design state-of-the-art, we propose a conceptual, information-exchange interface – BridgeSec – which introduces vulnerability management aspects into system design efforts and automates some of them. In this section, we first describe the conceptual interface that we identified as a bridge between the systems engineering and the security engineering perspectives and highlight the overarching approach and design of BridgeSec. Then, we formally introduce BridgeSec's concepts and the related reasoning mechanism. Finally, we present a prototype implementation of the proposed interface.

3.1. BridgeSec: conceptual design and description

To address the gap of systematically coordinating security aspects of the system between the systems engineering and the security engineering perspectives, it is necessary to identify an interface that allows to explicitly communicate these aspects between the disciplines. Such an interface should enable the separation of concerns that would allow each discipline to act within its realm of responsibility and employ the relevant disciplinary information and expertise.

While attempting to appease the two perspectives, and by inspecting the related security body of knowledge (Section 2), we identify such a conceptual information-exchange interface in the form of two concepts: component types and controls. Fig. 2 presents this newly unveiled interface in the context of the security by design effort. Systems engineering is the discipline that is responsible for specifying the required functionality of a system as well as the architecture of the system. Security engineering is the discipline that holds valuable information about attacks, vulnerabilities that may lead to them, and controls that may mitigate these.

The conceptual interface specifies that systems engineering is responsible for curating component types used in the system design (as derived from the functionality and architecture), in any appropriate level of abstraction, e.g., from a generic software component type through a more specific type of a software unto a specific version of the software to be used. Similarly, security engineering is responsible for curating vulnerabilities and controls in any appropriate level of abstraction, e.g., CWEs as conceptual vulnerability mechanisms and CVEs as specific, concrete types of vulnerabilities in software packages.

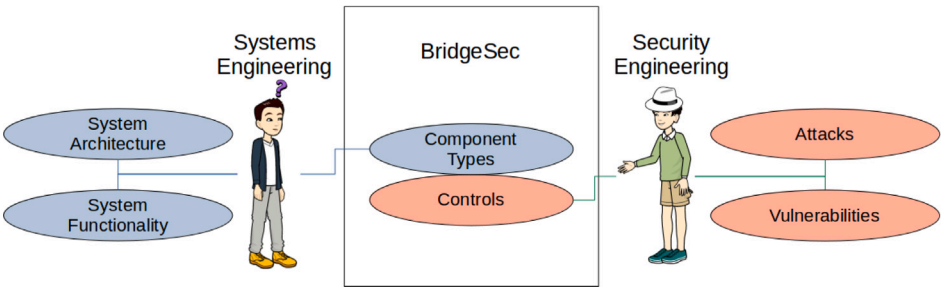

Fig. 2. BridgeSec interfaces the system engineer's view and the security expert's view.

Component types, as communicated by systems engineering, are analysed by security engineering to identify potential vulnerabilities (in the pertinent level of abstraction). Furthermore, security engineering is expected to propose controls to mitigate these vulnerabilities, corresponding with the level of abstraction of the component type. Systems engineering is informed of potential vulnerabilities associated with the various system components, based on their types and in the appropriate level of abstraction. The systems engineering team can also examine the level-appropriate controls offered by the security engineering team and embed them into the design specification, either as additional architectural elements (e.g., Firewall) or additional functionality (e.g., message filtering). This is in alignment with systems engineering practices of specifying system architecture as an arrangement of components, as well as specifying and managing required functions – and later requirements – at various levels of abstraction. Some of the functions/requirements management aspects include allocating the functions/requirements to the system components that should provide them, for further development by the component's development team [5].

To leverage the identified interface as an inter-disciplinary communication mechanism and to facilitate its use for designing systems for security, we propose the BridgeSec prototype implementation. The prototype allows systems engineers to depict system architecture and security-related functionalities as well as to type the various components used within the system design. It also allows security engineers to specify vulnerabilities and security controls. These are all currently done manually.

The typing of components – by systems engineers – and the specification of vulnerabilities and controls – by security experts – can be an iterative process, to which both disciplines contribute. For example, specific types can be further suggested by security experts in order to facilitate their understanding of components and – as a result – identification of potential vulnerabilities. In certain scenarios, security experts can also specify component types as part of an organisational policy of mandating mitigation for component types of interest. For example, a security team can mandate the use of a memory-safe programming language for software components that are internally developed. For this, the security team will need to define "internally developed software" as a component type, which will then be used by systems engineering to characterise every internally developed software item in the system design. Similarly, skilled systems engineers can offer mechanisms that may be considered as security controls to address different vulnerabilities, based on their domain expertise and/or experience with other, somewhat related design objectives such as safety and resilience. These controls can be added to the mitigation body of knowledge developed by security experts. Consequently, BridgeSec can support design decisions about the security posture of the system under development as well as on the security controls that need to be incorporated into the system, as detailed shortly.

Once definitions of system-related component types exist, security engineers can associate vulnerabilities with the component types that are prone to such vulnerabilities, as well as with security controls that can mitigate the vulnerabilities. Inspired by the terminology of previous vulnerability management work [19], we term such associations collectively as a definition of a vulnerable asset (VA). VA allows a security expert to describe a related set of vulnerabilities that may be exploited when a component of a specific type is under attack, potentially resulting in a security compromise. Also, the security expert is able to identify a set of controls that may mitigate the collective set of vulnerabilities. As a design decision, our current BridgeSec implementation treats each of the associated controls – individually – as proper potential mitigation. If an expert wishes to mandate the collective use of several security controls as mitigation, he/she can create multiple vulnerable asset definitions, with each single definition relating to a specific control from the mandated set.

Fig. 3 illustrates the use of BridgeSec as a process framework. It identifies key activities and data objects (artefacts) that can be used to systematically reason about the security of system designs and improve them. The general flow of the activities depicted in the figure is top to bottom, although, realistically, we expect iterative applications of this flow. Security experts may specify component types (the "Specify component types" activity) to result in definitions of component types that can be used for system design, e.g., in support of an organisational security policy. System engineers are expected to type the components that appear in the requirements specification and architecture ("Type components" activity). For this, systems engineers can use component types previously defined by security experts and/or further contribute new component types to the organisational knowledge base. The "Type components" activity should also result in updates to the requirement specifications and/or architectures (e.g., by incorporating identified component types as metadata/attributes for components in system models).

The creation of component types may involve the use of information from existing knowledge bases. Whenever new component types appear in the Component Types collection, security experts are expected to identify vulnerabilities that may affect components of this type (the "Identify/specify vulnerabilities" activity, resulting in an updated collection of vulnerabilities). This can be the result of researching attack and vulnerability knowledge bases. Security experts then proceed to curate the existing knowledge in the form of vulnerable assets (the "Specify vulnerable assets and security controls" activity). They attribute vulnerabilities (from the Vulnerabilities collection) to pertinent component types (from the Component Types collection) resulting in vulnerable asset definitions. In these definitions, they also identify pertinent security controls, from the existing Security Controls collection. If new controls are needed, they are added to the Security Controls collection. The collection can be populated by importing information from existing knowledge bases (such as NIST SP 800-53).

Finally, system engineers can project vulnerable asset definitions on their design (artefacts such as requirement specification and architecture, which already include component type associations), and assess which components are vulnerable. For the vulnerable components, system engineers should then design the mitigation, by adopting controls (from the Security Controls collection), adapting them or introducing new security controls. While the proposed scheme clearly outlines the

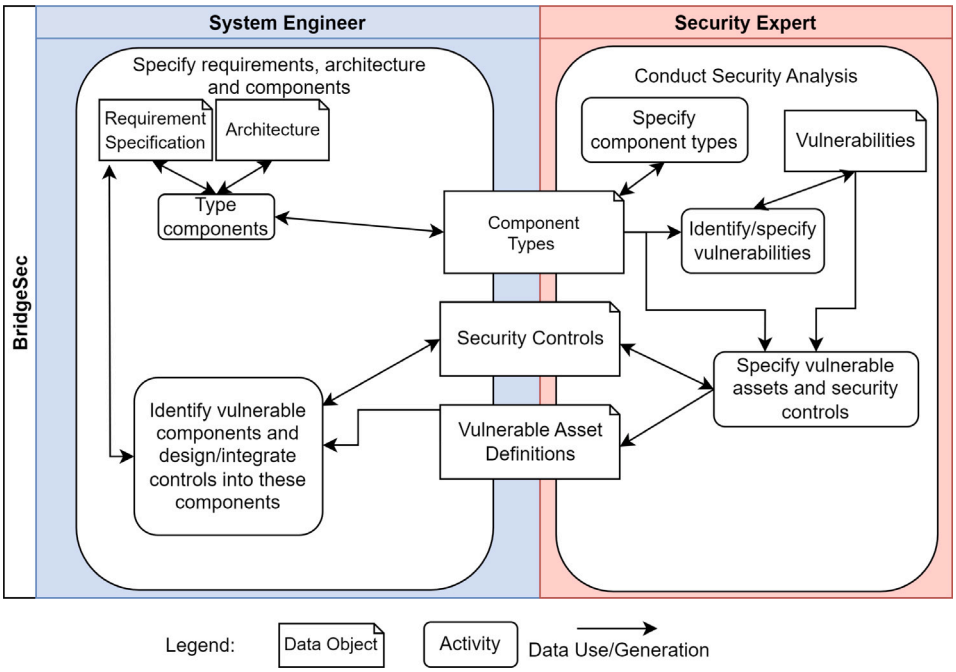

Fig. 3. System engineer's and security expert's activities conducted in the scope of BridgeSec.

responsibility of each discipline and delegates each discipline to lead specific activities, all activities may include acts of consultation and coordination between the disciplines.

Additionally, some of the activities may be automated. BridgeSec incorporates a reasoning mechanism that infers whether system components (specified in the system architecture description) are vulnerable, based on VA definitions and the controls associated with the components. This automates part of the "Identify vulnerable components and design/integrate controls into these components" activity of Fig. 3. The automated reasoning mechanism is applied individually for every component of the system. The mechanism:

- collects all predefined VAs that can be relevant to the specific component, based on its predefined component types;
- checks if the controls currently associated with the specific component provide potential mitigation for the entire collection of VAs, by making sure that each VA is covered at least by one of the controls designed to be integrated into the component. If there is one or more VAs left unmitigated (by the potential design), then the reasoning mechanism identifies the component as vulnerable.

We provide the formal description of VA as well as the reasoning mechanism in the next subsection.

3.2. BridgeSec: formal description

BridgeSec provides a semi-automated capability to identify vulnerable components in the design of systems. The capability incorporates two mechanisms: (1) formal definition of vulnerable assets (Section 3.2.1, manual), and (2) vulnerable component assessment (Section 3.2.2, automatic). Next, we detail these two mechanisms.

3.2.1. Formal definition of vulnerable assets

The construction of a VA definition in BridgeSec is a manual mechanism, designed to be used by a security expert and to capture security-related knowledge and/or policy. A vulnerable asset (VA) binds a specific type of component with one or more vulnerabilities. This is the mechanism that allows the expert to express knowledge about vulnerabilities that pertain to a specific type of component. The VA may also bind security controls that can, potentially, mitigate the associated vulnerabilities. This mechanism does not depend on a specific system design. Theoretically, it requires no input from system designers. However, the VAs should be developed with the relevant system domain in mind.

Formally, a VA is the tuple of a set of component types, a set of vulnerabilities, and a set of security controls as specified in Eq. (1):

$$\mathrm{VA} \equiv \{\mathrm{ComponentTypes}, \mathrm{Vulnerabilities}, \mathrm{Controls}\} \tag{1}$$

where:
ComponentTypes is a set of specific component types;
Vulnerabilities is a set of vulnerabilities;
Controls is a set of security controls, of which each element may mitigate the Vulnerabilities.

3.2.2. Automated reasoning for vulnerable component assessment

The reasoning mechanism allows to automatically assess whether a specific component is a vulnerable component. For a specific component, all associated component types are collected to compose the set of applicable component types (ACT). We note that a component may be associated with any number of component types, to provide designers the freedom to express components as well as component types in any hierarchy or conceptual level. Then, from the set of applicable component types, we retrieve all VAs associated with every component type. This composes the set of applicable VAs (AVA). Then, we define the set of all security controls currently assigned to the specific component (ASC). Finally, we check if the currently assigned security controls include at least one of the security controls associated with every applicable VA. If the result is false, then the component is assessed vulnerable, i.e., it is subject to one or more vulnerabilities associated with at least one VA.

Formally, we define the following sets:

$$\begin{aligned}&\mathrm{ACT}(\mathrm{component}) \equiv \\ &\{\mathrm{type} | (\mathrm{type} \in \mathrm{BridgeSecComponentTypes}) \\ &\wedge (\mathrm{type} \in \mathrm{ComponentTypes}(\mathrm{component}))\}\end{aligned} \tag{2}$$

where:
ACT is the set of applicable component types for a specific component;

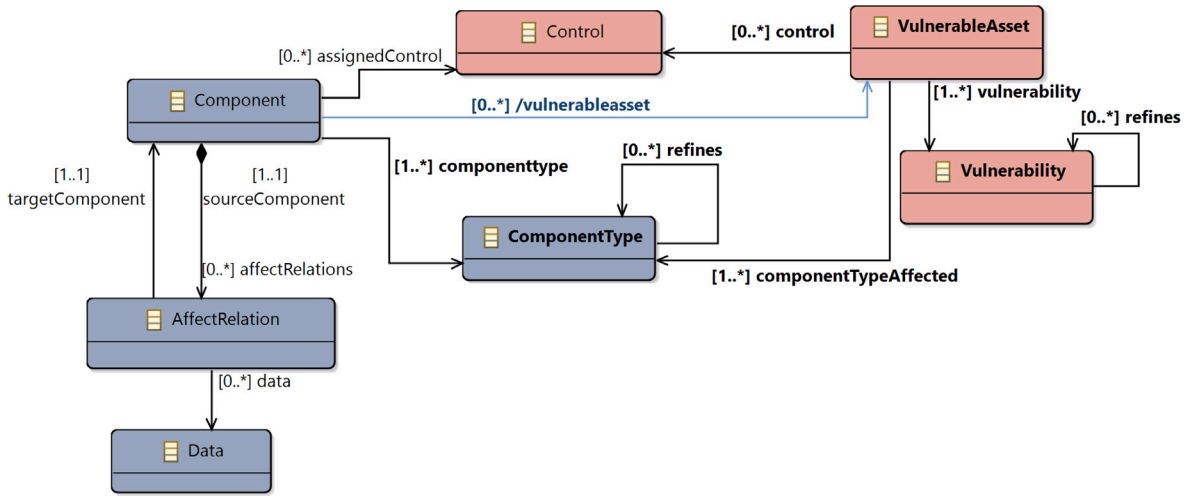

Fig. 4. BridgeSec Metamodel. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

BridgeSecComponentTypes is the set of all predefined component types in BridgeSec;
ComponentTypes is the set of all predefined component types associated with a specific component.

$$\text{AVA}(\text{component}) \equiv \{\text{VA} \mid \text{VA} \in \text{BridgeSecVAs} \wedge \text{VA.ComponentType} \in \text{ACT}(\text{component})\} \quad (3)$$

where:
AVA is the set of applicable VAs for a specific component;
BridgeSecVAs is the set of all predefined VAs in BridgeSec.

Then, our mechanism uses the above definitions to assert if a specific component is vulnerable or not, based on the following logic:

$$\neg\Big(\forall \text{VA} \in \text{AVA}(\text{component}), \exists \text{control} \in \text{ASC}(\text{component}) \wedge \text{control} \in \text{VA.Controls}\Big) \rightarrow \text{VC}(\text{component}) \quad (4)$$

where:
¬ is the symbol for negation (“not”);
ASC is the set of security controls currently assigned to a specific component;
VA.Controls is the set of security controls included in the definition of a specific VA (as potential mitigation);
→ is the symbol for inference/implication (“implies”);
VC indicates if a specific component is vulnerable.

In our design, a single security control of the set of potential mitigations is sufficient in assessing that a component is no longer vulnerable. This supports a selection of controls between various options. Our design, however, also provides an option for the security expert to express more complex situations and/or policies – i.e., policies that require a combination of security controls – by introducing multiple VAs. The inference about not being a vulnerable component is in the form of conjunction with respect to the entire set of VAs, i.e., unless a component has at least one control for every associated VA, it is assessed as vulnerable.

This automated reasoning can be applied to a component at any level of the design hierarchy. It is a preliminary design tool. It only relates to the potential of mitigating vulnerabilities. Specific designs and/or configurations should be more specific with respect to how the controls are effectively integrated into the component, e.g., by following rigorous requirements engineering processes and techniques [86], and, particularly, by specifying vulnerabilities and security controls in the proper level of abstraction and in association with a suitable level of the system design hierarchy.

3.3. BridgeSec prototype implementation

Based on the above formal description, we developed a prototype implementation of BridgeSec. Utilising a model-driven approach, the implementation includes a formal, executable metamodel and dynamic representations.

The prototype implementation is available as an EPL 2.0-licensed open-source package at https://github.com/IAI-Cyber/TRADES/releases/tag/v3.5-beta. The implementation has been fully integrated into the TRADES Tool online repository [44]. We chose TRADES Tool as the platform for the implementation as it is open-source and rigorously relies on an explicit metamodel for detailing security-related models and representations. Furthermore, some of the BridgeSec concepts – specifically *component* and *control* – were already supported, and this facilitated our implementation.

3.3.1. The prototype’s metamodel

Fig. 4 shows the prototype’s metamodel, which includes the BridgeSec system’s concepts and their relations. The metamodel extends the previously developed TRADES metamodel concepts and relations (normal font) with new concepts and relations appearing in bold. Concepts are depicted as nodes (boxes), and the relations between these concepts are depicted as edges between the nodes. A relation might have specific cardinality, which is explicitly mentioned in square brackets ([1..1] represents a single element is required as the associated end of the relation, [0..*] represents any number of elements is allowed, and [1..*] represents one or more elements are required as the associated end of the relation).

The metamodel incorporates various concepts associated with different perspectives, represented using distinct colours strictly for clarity purposes: vulnerability management perspective (red) includes Control, Vulnerable Asset, and Vulnerability, while the system perspective (blue) comprises Component, Component Type, Affect Relation, and Data.

The system-related concepts support the detailing of the system to the appropriate level. The “Component” concept is used to represent the system and its constituents. Each component can have one or several component types. We use the “refines” relation to allow modelling different levels of abstraction, providing support for the hierarchical organisation of the type system. This is denoted using a “refines” relation from the Component Type concept to itself. A component may have any number of associated controls as its features/requirements, denoted by the “assignedControl” relation from the Component concept to the Control concept. Based on previous TRADES design, Affect relations can be depicted between components (the “affectRelations”

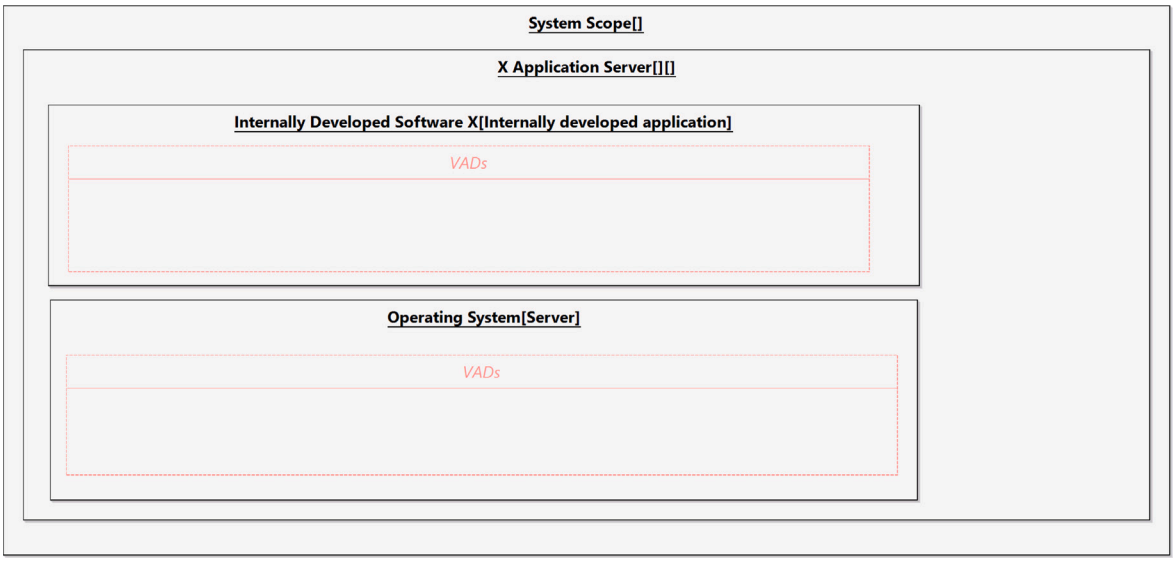

Fig. 5. Scenario I system design representation using BridgeSec prototype. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

composition relation from the source "Component" to "AffectRelation"; and the "targetComponent" reference relation from "AffectRelation" to the target "Component"). The affect relations are directional data flows, and each may convey any number of data elements (the "data" reference relation to "Data" concept).

The vulnerability-management-related concepts facilitate the introduction and management of vulnerability information. The "Vulnerability" concept is used to represent the vulnerabilities. A lower-level vulnerability can refine a higher-level vulnerability (e.g., a CVE vulnerability can refine a CWE vulnerability) using the "refines" relation. The Vulnerable Asset concept is the core conceptual element of the reasoning mechanism. As described and formally specified in the previous subsection, the Vulnerable Asset relates to one or more vulnerabilities, associated with one or more component types and potentially including suggested controls. The derived "vulnerableasset" relation from Component to Vulnerable Asset (annotated in blue) represents our reasoning mechanism's inference that a component is subject to a specific vulnerable asset definition.

In the BridgeSec prototype, models of systems – that are to be designed or analysed – are created exclusively based on the metamodel, i.e., every model element is an instance of a metamodel element. By specifying the concepts and relations using a well-structured, executable metamodel, we provide a rigorous, comprehensive framework for modelling and analysing system components, component types, vulnerabilities, controls, and their interconnections, supporting vulnerability management and the system design for security.

3.3.2. The prototype representation

The prototype representation is a model-based representation, i.e., it is dynamically generated based on the content of an information model according to rules defined with respect to the metamodel. Examples of generated representations appear in the following section (see Figs. 5–14). The representation extends the TRADES design diagram representation, by adding: (1) a list of types associated with every component in square brackets; (2) a Vulnerable Asset Definitions container (named "VAD" in the figures), which is placed within the scope of system components and names and provides access to applicable VAs (AVA) associated with the component by the reasoning mechanism (see Eq. (3)); and (3) a red colouring of the label and border of the box representing a specific component, in case the reasoning mechanism infers that the component is vulnerable (see Eq. (4) for the reasoning specification and Fig. 6 as an example).

The reasoning mechanism queries are implemented using the Acceleo Query Language (AQL), based on the formal specification (Section 3.2.2). AQL supports evaluating basic predicate logic statements over sets of model elements.

4. Illustrative scenarios

To demonstrate the applicability of BridgeSec, we illustrate its use in exemplary scenarios. These scenarios are representative of prominent security by design challenges related to managing and mitigating vulnerabilities [13] as well as of the potential applications of BridgeSec in real-world scenarios to address these challenges.

The first scenario demonstrates how BridgeSec can be applied in the early phases of development to manage information related to systems security. Specifically, it describes the use of BridgeSec as a mechanism to explicitly state security policies and coordinate them with the system designers; and, in turn, the use of BridgeSec to support the inclusion of security controls in the system design and in the development process. This is described using a realistic scenario.

The second scenario depicts a characteristic application of BridgeSec to coordinate the necessary modification of systems – after specific components are selected or implemented and even during maintenance – due to the disclosure of vulnerability information. This scenario includes two stages, to further illustrate the use of BridgeSec throughout the system lifecycle. Since disclosure of new vulnerabilities is unpredictable, it is not realistic to evaluate new solutions while waiting for such disclosures. Instead, we use past information and present a "what-if" implementation that treats previously disclosed vulnerabilities as if they are new information that needs consideration. Specifically, this scenario shows how BridgeSec can support informed decision-making to address several vulnerabilities that exist in system components and are revealed gradually over time.

The full models for both scenarios are available in the TRADES open-source repository (https://github.com/IAI-Cyber/TRADES/tree/master/examples/BridgeSec_WS).

4.1. Scenario I: Coordinating security features

We describe an illustrative scenario of using BridgeSec for coordinating desirable security features of the system during the requirements definition and analysis and the design phases of the system development life cycle.

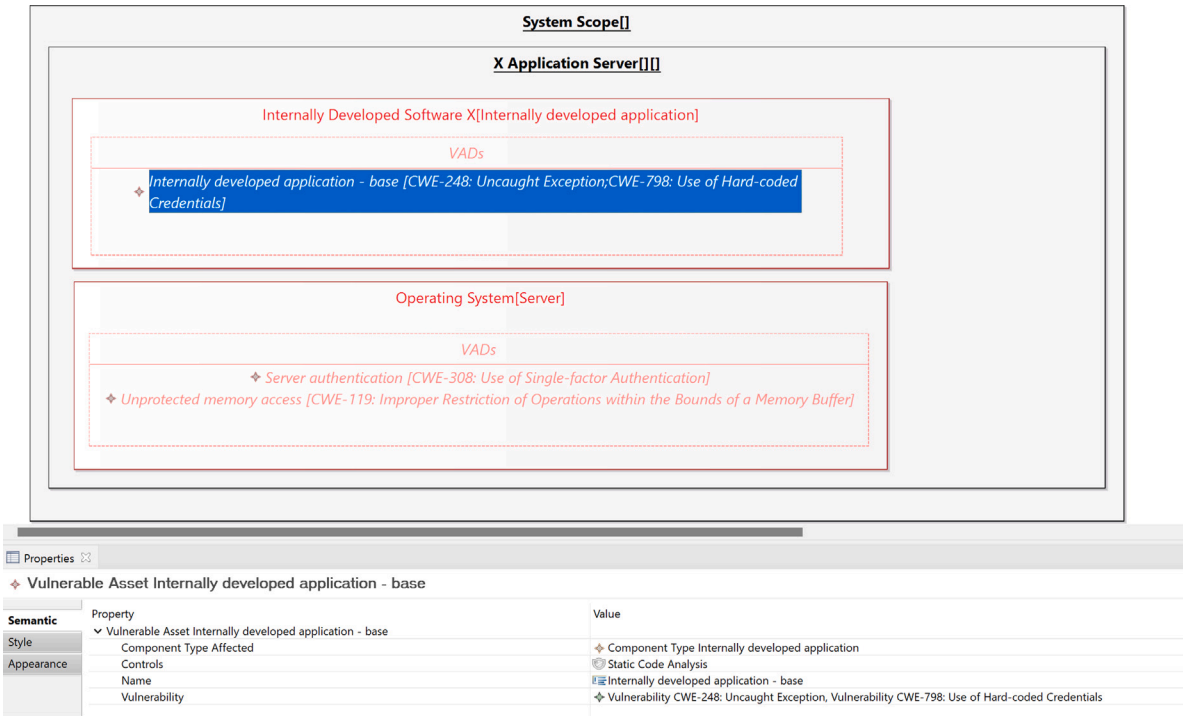

Fig. 6. Scenario I system design representation, showing the security posture. Applicable VAs are associated with components, inferred as vulnerable, hence appearing in red. Proposed security controls can be identified using the properties of each VA. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Consider the development of a server-side application within an organisation, designated for use by the organisation, its clients, partners, and subcontractors. This is a realistic situation for many enterprises that rely on information technology (IT) for their operations. We view the full application – X Application Server – as consisting of: (a) an operating system for server applications; (b) an internally developed server software. The development team seeks guidance for security mechanisms that are required or desirable as part of the system's design.

The types for the two components are specified by the development team as "Server" and "Internally developed application" for the operating system and for the internally developed software respectively. Such type definitions may be standardised by the organisation or a community of interest. Lower-level definitions may be specified instead or in addition to such high-level definitions at any stage of the development, if the specific technology is a constraint or a design decision. For example, the server in our scenario can be later specified as a "Windows Server" typed component, if the development team chooses to use a Windows server or if the organisational policy requires Windows servers to be used exclusively. Upon selection of a specific Windows Server version, this can also be specified as a type, e.g., "Windows Server 2019". Fig. 5 shows the resulting representation of the design using the BridgeSec prototype.

A security expert specifies vulnerable assets (VAs) pertinent to the design. These VAs may be based on organisational policies, e.g., mandating each internally developed application to undergo specific verification processes. They can also be in response to a new type definition specified by the development team, e.g., when selecting a component that has never been used by the organisation before. In our scenario, the security expert specifies three vulnerable assets: VA1 (named "Server authentication"), VA2 (named "Unprotected memory access"), and VA3 (named "Internally developed application - base"), as shown in Fig. 6 and as specified in the following set definitions:

$$\text{VA1} \equiv \Big\{ \{Server\}, \{\textit{CWE-308: Use of Single-factor Authentication}\}, \{\textit{ia-2.2: Multi factor Authentication to Non privileged Accounts}\} \Big\} \tag{5}$$

$$\text{VA2} \equiv \Big\{ \{Server\}, \{\textit{CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer}\}, \{\textit{si-16: Memory Protection}\} \Big\} \tag{6}$$

$$\text{VA3} \equiv \Big\{ \{\textit{Internally developed application}\}, \{\textit{CWE-248: Uncaught Exception, CWE-798: Use of Hard-coded Credentials}\}, \{\textit{sa-11.1: Static Code Analysis}\} \Big\} \tag{7}$$

In the specific case, the vulnerabilities associated with these VAs use publicly available information about vulnerabilities, by referring to specific CWE records. It is noted that VA3 binds two CWEs that are applicable to the same component type and can be mitigated using the same control. Similarly, the security controls associated with these VAs use publicly specified security control definitions, by identifying specific NIST SP 800-53 controls, with the prefix of each control denoting its NIST SP 800-53 unique identifier [48]. While the VA definitions use publicly available information, the integration of this information – specifically the potential mitigation of specific vulnerabilities by specific security controls – as VAs is a codification of the security expert's knowledge and/or of organisational policies.

Fig. 6 shows the security posture of the system design. Both system components are automatically coloured red, based on the vulnerable component inference of the BridgeSec automated reasoning mechanism. The names of the VAs (assessed as the cause for a component being vulnerable by the reasoning mechanism) appear explicitly within the "VADs" container in each component box, alongside the vulnerabilities associated with the VA in brackets. As a side note, the graphical user interface element that represents a single VA allows the user to

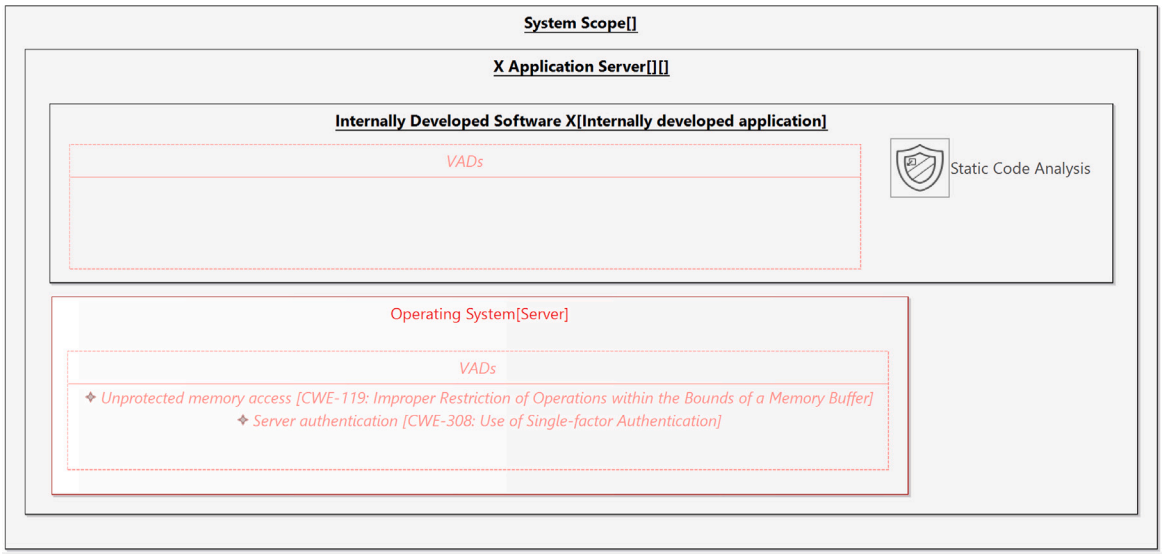

Fig. 7. The system security posture is updated with a design decision to incorporate the "static code analysis" security control. This reflects the internally developed component is no longer vulnerable, appearing in black (and not red). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

select the VA and see the properties related to the VA. Fig. 6 shows the "Internally developed application - base" VA selected and, accordingly, its properties appear in the bottom "Properties" panel.

The systems engineering team is now made aware that the design is vulnerable and/or does not comply with security policies. The developers may also probe the VAs to examine proposed security controls to address specific VAs. The properties panel in Fig. 6 shows that the "Internally developed application - base" VA can be addressed by including "Static Code Analysis" (listed at the relevant "Controls" field in the Properties panel). This illustrates the mechanism for communicating vulnerabilities and expected security controls between system engineers and security experts.

Based on the VAs specified by the security expert, the systems engineering team can now design relevant security aspects of the system and its development. Specifically, the team can make an informed decision to incorporate static code analysis – a security control suggested by the security expert – into the development process. Fig. 7 shows the resulting update to the security posture, once the "Static code analysis" security control is incorporated into the scope of the Internally Developed Software X component. The component no longer appears in red (it is now coloured black), as the inference mechanism did not find it to be vulnerable.

The same approach applies to the "Operating System" component. First, the "memory protection" security control can be incorporated to mitigate the "Unprotect memory access" VA, as Fig. 8 shows. Then, the "Multi-factor authentication to non-privileged accounts" control can be incorporated to mitigate the "Server authentication" VA, as Fig. 9 shows; resulting in the removal of the "vulnerable" tag from the "Operating system" component. We note that the existence of a single VA associated by inference with the component is sufficient for the component to be classified as vulnerable. This is in full accordance with the formalism that appears in Section 3.

These representations allow the security expert to assess the security posture of the design at any time and clearly understand how security controls are embedded into the design. Moreover, by associating security controls with specific system components, it becomes evident who holds the responsibility for implementing these controls: the development team of the specific component. This provides clarity and transparency regarding the incorporation of security measures, enabling their effective management throughout the development process.

4.2. Scenario II: Addressing implementation-level vulnerabilities

We describe another illustrative scenario, using BridgeSec for vulnerability management during the maintenance phase of the system development life cycle. This scenario is also indicative of how BridgeSec can be employed for addressing security concerns in middle-out systems engineering, when the system design incorporates existing lower-level components. The system of interest is an extended view of the system which appears in Scenario I (previous subsection), with the addition of other components to illustrate multiple development efforts and deployment considerations (Fig. 10). Specifically, the system comprises a server-side application in its deployed form – designated "Application X Server"; as well as a client-side application (which interacts with the server-side application) in its deployed form – designated "Application X Client". Component types are identified with their CPE identifications, appearing in brackets in the title of each component box. The identification of the Application X Server's Operating System – as a Windows Server 2019 CPE entry – is a refinement of the "Windows Server" component type, which is itself a refinement of the "Server" component type. These refinements demonstrate the ability of systems engineering to detail a component type to the required level of abstraction as design decisions are made throughout the development process.

Consider a disclosure of a new vulnerability in Microsoft Remote Desktop Services (RDS) implementation. Real-life vulnerabilities representing this compose the set of vulnerabilities published on August 14, 2019: CVE-2019-1181 [87], CVE-2019-1182 [88], CVE-2019-1222 [89] and CVE-2019-1226 [90]. All these CVEs are rated *Critical* and relevant to various Windows platforms, including Windows Server 2019 and Windows 10 (which are both used in our system design). The vulnerable component types are identified using their CPE enumerations on the CVEs' "Known Affected Software Configurations" section. The proposed mitigation for each of these vulnerabilities, as offered by the vendor advisories (see, for example, [87]), is to apply a security patch, or to disable the specific services (RDS) if they are not required. Applying the relevant security patch can be identified as a specific refinement of the NIST SI-2 "Flaw Remediation" security control, which necessitates to "install security-relevant software and firmware updates". Disabling RDS can be identified as a specific refinement of the CM-7(1) "Least Functionality", clause (b) "Disable or remove".

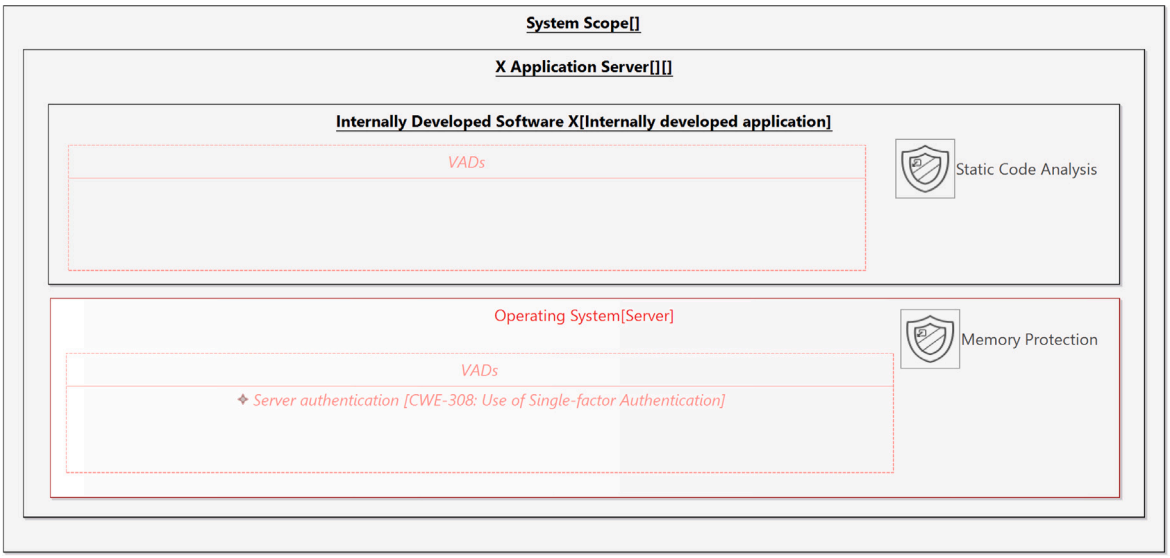

Fig. 8. The system security posture is updated with a design decision to incorporate the "memory protection" security control. This reflects the operating system component is no longer vulnerable to the specifically addressed VA, but it remains vulnerable to another VA, hence still appearing in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

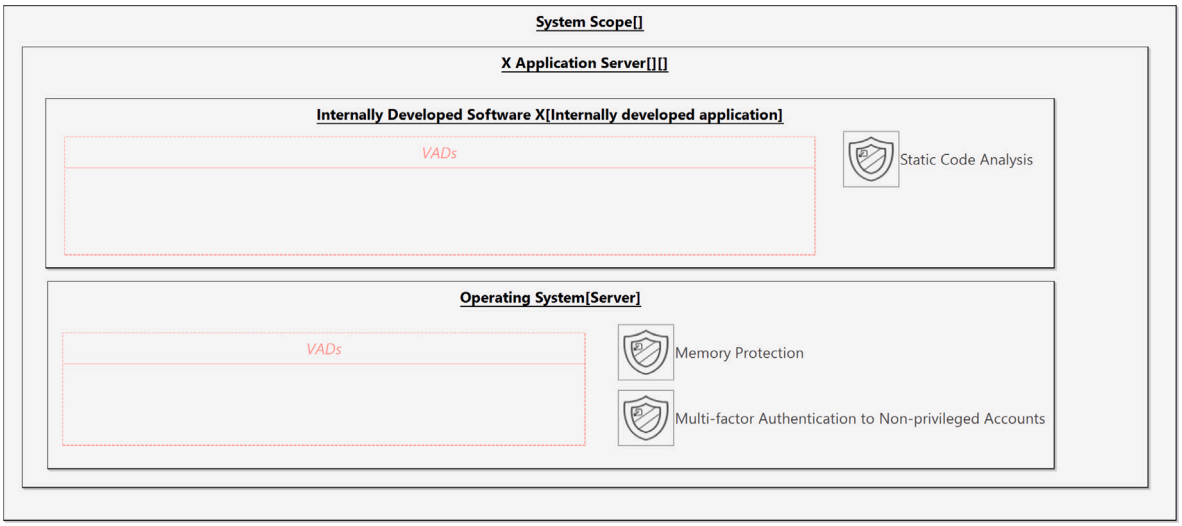

Fig. 9. The system security posture is updated with a design decision to incorporate the "Multi-factor authentication..." security control. All components appear in black, signifying no component is vulnerable according to the current design. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Following the information provided above, a security expert specifies a new vulnerable asset, VA4, named "Remote Code Execution via RDP Server":

$$\begin{aligned} \text{VA4} \equiv \Big\{ & \{\text{cpe} : 2.3 : \text{o} : \text{microsoft} : \text{windows\_10\_1507}, \\ & \text{cpe} : 2.3 : \text{o} : \text{microsoft} : \text{windows\_server\_2019}\}, \\ & \{\text{CVE} - 2019 - 1181, \text{CVE} - 2019 - 1182, \\ & \text{CVE} - 2019 - 1222, \text{CVE} - 2019 - 1226\}, \\ & \{\text{'WindowsRDPServerSecurityPatchAugust2019'}, \text{'DisableRDS'}\} \Big\} \end{aligned} \tag{8}$$

Our BridgeSec prototype then infers that two components are vulnerable: the Application X Server's operating system and the Application X Client's operating system. This is based on the types of components – the CPE enumerations of the specific Windows Server 2019 (cpe:2.3:o:microsoft:windows_server_2019) and Windows 10 versions (cpe:2.3:o:microsoft:windows_10_1507) – as specified, respectively, for the components by the system designers. Fig. 11 is the pertinent representation by the BridgeSec prototype, showing the vulnerable components outlined in red.

The new vulnerability-related issues are communicated with the systems engineering team, using BridgeSec. Now, the systems engineering team is aware of the vulnerable components and considers the suggested security controls in light of the system functionality (Fig. 2). Since RDS is not a required functionality of the Application X Client, the team decides to disable RDS in the client. Also, since the RDS relates to the functionality required by the Application X Server (e.g., for maintenance purposes), applying the vendor's security patch is selected as the security control for the server's operating system. Fig. 12 shows the design of such actions, which is, in fact, the revised design of the system in its new version/deployment; and the resulting security posture analysis, showing both components are no longer considered vulnerable.

Now, consider the disclosure of another vulnerability in Microsoft Remote Desktop Services at a later point in time. One such example is the disclosure of the CVE-2020-0655 vulnerability on February 11,

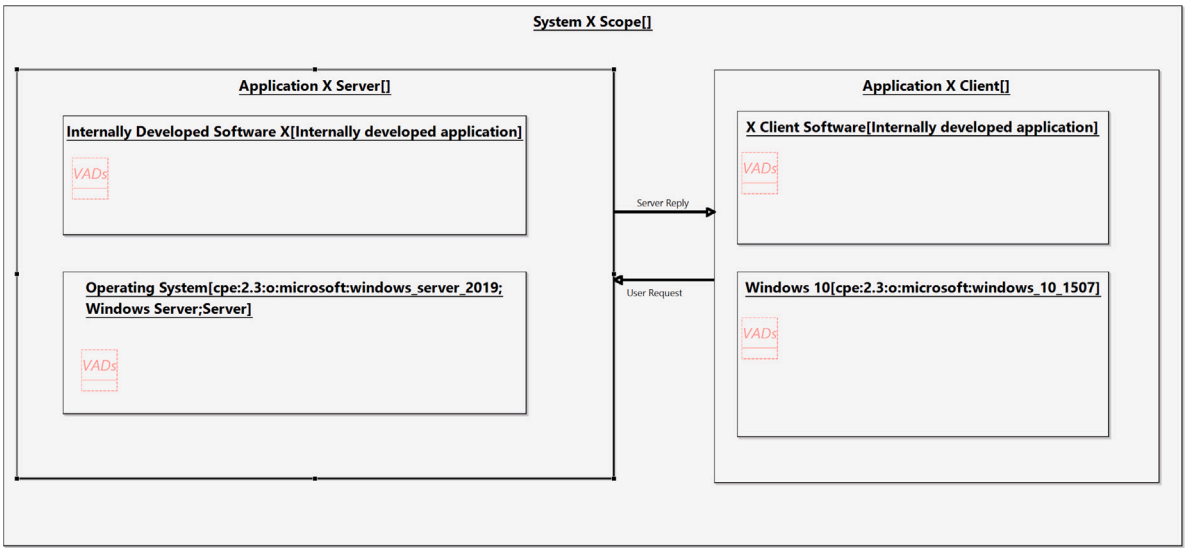

Fig. 10. Scenario II system design and the initial security posture, shown using the BridgeSec prototype. As a starting point, none of the components is inferred as vulnerable. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

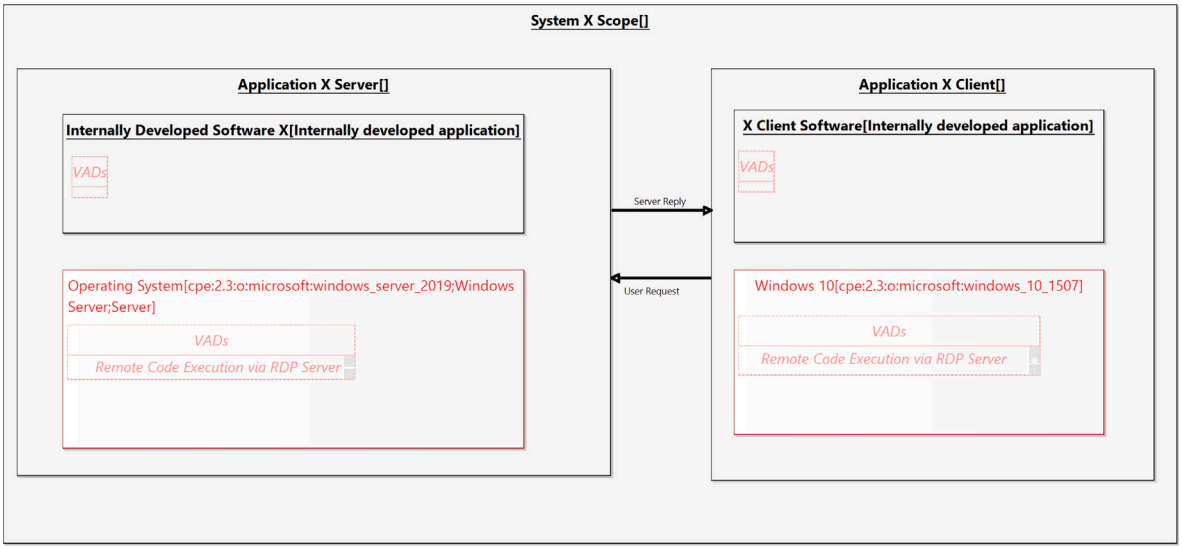

Fig. 11. Scenario II updated system security posture, following a new VA definition after a new set of vulnerabilities were disclosed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

2020 [91]. This CVE is rated *High* and relevant to various Windows platforms, including Windows Server 2019 and Windows 10. As a side note, unlike the advisories for the previously mentioned vulnerabilities, the vendor advisory for this vulnerability does not include a "mitigation" section [91]. The proposed mitigation – installing a security update – is implicit from the "Executive Summary" section. The other alternative, disabling RDS, can also be inferred from the description in that section but requires a more technical orientation from the reader. This provides further evidence for the informal nature of current knowledge bases; and it attests to the gap addressed by BridgeSec, as discussed shortly.

According to this new information, a security expert specifies a new vulnerable asset VA5, named "Remote Code Execution via RDP Server 2020":

$$\begin{aligned} \text{VA5} \equiv \Big\{ & \{\text{cpe} : 2.3 : \text{o} : \text{microsoft} : \text{windows\_10\_1507}, \\ & \text{cpe} : 2.3 : \text{o} : \text{microsoft} : \text{windows\_server\_2019}\}, \\ & \{\text{CVE} - 2020 - 0655\}, \\ & \{\text{'WindowsRDPServerSecurityPatchAugust2020'}, \text{'DisableRDS'}\} \Big\} \end{aligned} \tag{9}$$

Fig. 13 shows the inference results made by the BridgeSec prototype implementation. The prototype infers that the Windows Server's operating system of our application system is vulnerable. It is noteworthy that the application client's operating system is not assessed as vulnerable, because one of the controls associated with VA5 – "Disable RDS" – has already been integrated into the component when responding to the previously raised VA4. The vulnerability in the application server's operating system can be mitigated by applying the new security patch, as Fig. 14 illustrates.

5. Related work

Our work establishes a conceptual and explicit information-exchange interface between systems engineering and security engineering, to articulate the design for security of systems. While various security-related conceptualisations exist, we are unaware of any conceptualisation designed to support information exchange between disciplines during system development. Avizienis et al. introduce a taxonomy of dependable and secure computing, without offering a

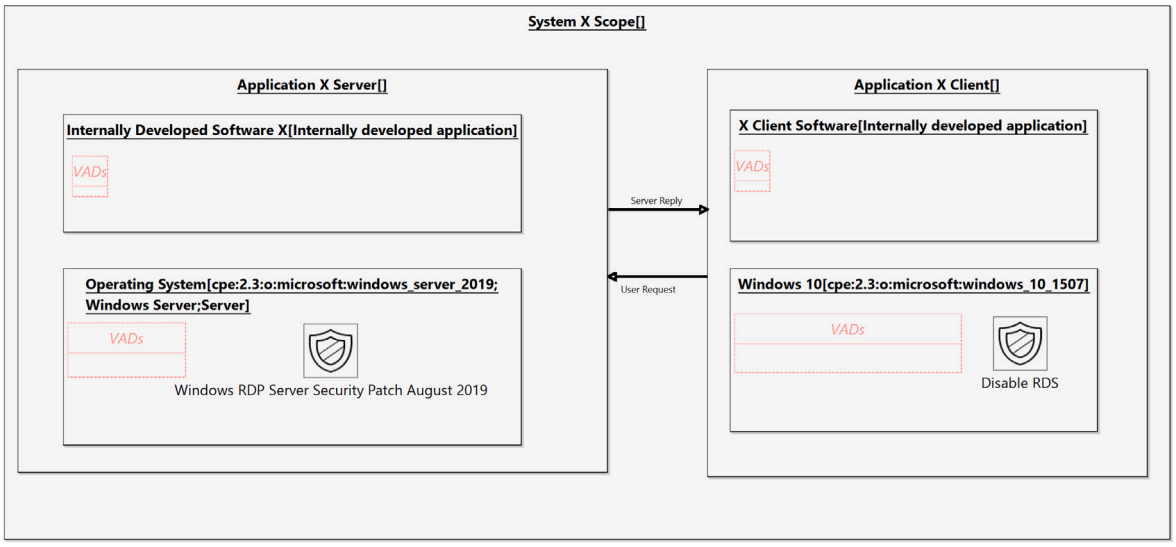

Fig. 12. Scenario II updated system security posture, after pertinent security controls are added to address the newly identified VAs. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

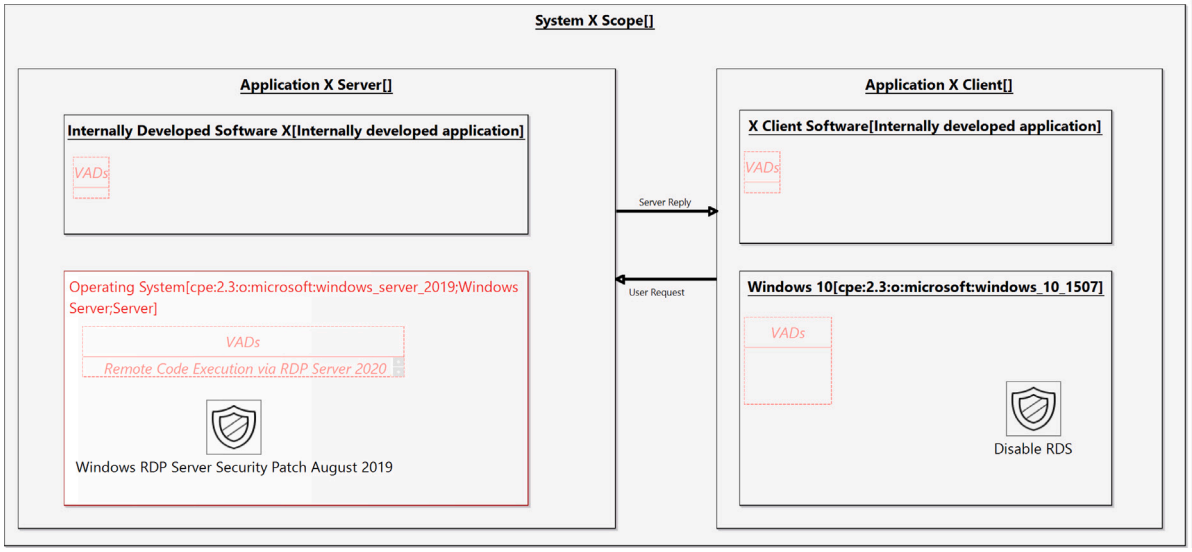

Fig. 13. Scenario II updated system security posture, after another vulnerability was disclosed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

validated approach to exercise the taxonomy's concepts in systems development [92]. Fenz and Ekelhart present an ontology that focuses on security concepts in support of risk management [93]. Oliveira et al. offer a formal ontology that integrates aspects of risk management and security engineering primarily for supporting risk treatment [94]. These publications [92–94] fall short of demonstrating explicit methods to integrate security into systems engineering processes utilising the specified concepts. Accordingly, they fail to address the gap in strategic technical management approaches for security, which is highlighted in the recent study by Olivero et al. [36].

In comparison, our work identifies specific concepts and establishes their systematic use within the development process, in the form of an information-exchange interface. This interface is detailed and formalised to a degree where semi-automated reasoning can be applied, as demonstrated by our prototype implementation. Our work concretely addresses the pertinent future work questions posed by Olivero et al.

Furthermore, while previous works successfully identified concepts similar to those used within BridgeSec (specifically, Control, Attack and Vulnerability, as shown in Fig. 2), only a few of them identify the need and demonstrate the ability to associate vulnerabilities with components based on component types/characteristics [93,72,28]. We establish Component Type as a focal concept in vulnerability management, as it underpins a by-design, divide-and-conquer strategy.

Additionally, our work corresponds with several related works that offer formal reasoning or model-based approaches to security. OnToRisk uses a formal ontology to assess vulnerability-induced risks [61]. The method requires formal ontology engineering proficiency and does not support the system design perspective. It provides a complementary, risk management perspective, which can be used to assess the business impact of a newly disclosed CVE-typed vulnerability once a system has been deployed.

Valenza et al. [33] propose a security analysis technique for systems. Like BridgeSec, their technique is formally defined. However, it does not accommodate the specification of vulnerabilities. It is oriented towards creating threat models and uses information about an entity being vulnerable to a threat, as opposed to BridgeSec, which allows specifying the nature of the vulnerabilities explicitly.

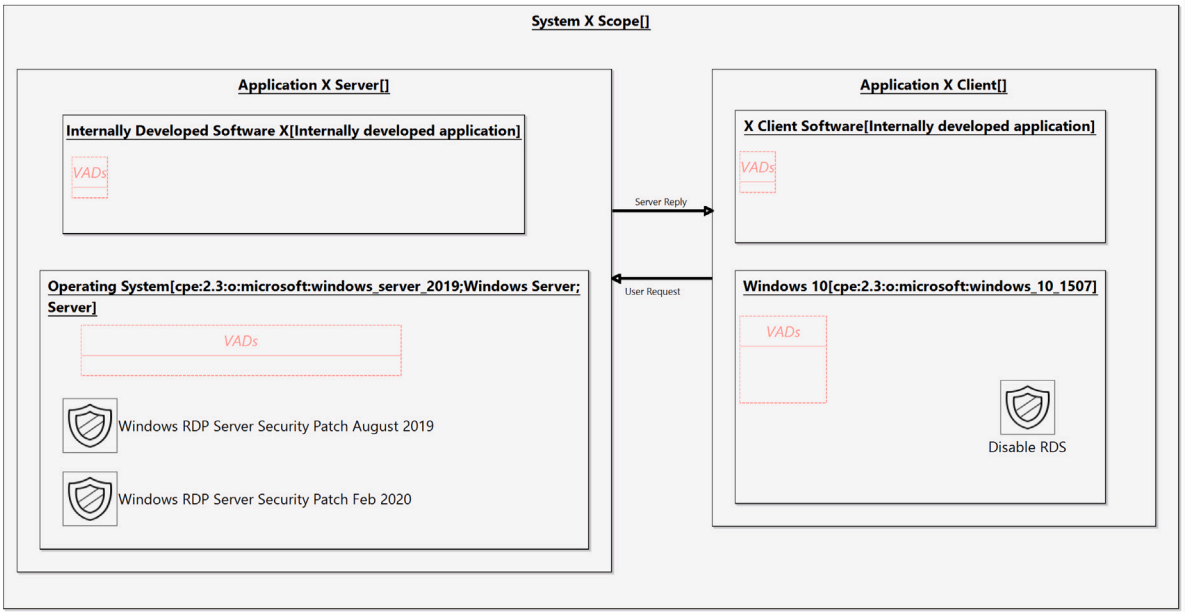

Fig. 14. Scenario II updated system security posture, after pertinent security controls are added to address the newly identified VA. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Eckhart et al. [95] suggest an ontology-based method that generates attack graphs, which are a generalisation of attack trees. Their method allows the automatic association of vulnerabilities with components, but does not take into account controls embedded in the system design. Adapting their method to address specific needs or contexts of the design requires extensive ontology-related knowledge and skills. In comparison, BridgeSec provides a simple mechanism to do so, by including the vulnerable asset concept as a binding definition for multiple concepts and applying reasoning with respect to vulnerable asset definitions. Their analysis results, as represented by the generated attack graphs, are not fed back into the system's design, whereas our prototype clearly shows how BridgeSec analysis output can be presented as part of the system design representation.

Tropos is a software engineering methodology that emphasises the identification and analysis of goals, actors, and dependencies in the early stages of system development via semi-formal modelling. Secure Tropos is an extension of Tropos to address security concerns including analysis of the security needs of the stakeholders and the system in terms of security constraints, identification of secure entities that guarantee the satisfaction of the security constraints, and assignment of capabilities to the system to help towards the satisfaction of the secure entities [96]. Contrary to Secure Tropos, which is primarily a goal-oriented requirements engineering methodology, BridgeSec focuses specifically on vulnerability management and design in the context of secure system development. As opposed to BridgeSec, Secure Tropos does not accommodate the existing security body of knowledge.

The VERDICT framework partially uses a model-based approach to capture security-related features of a system and then applies ontology-based rules for generating attack-defence tree representations [72]. However, VERDICT is designed as a tool exclusively for systems engineering. The security engineering perspective is represented by analysis rules that are hard-coded. These rules take into consideration only a specific subset of CWE records as vulnerabilities and a limited selection of security controls as well as very generic component types. VERDICT provides no flexibility to represent security-related policies, and it does not account for a proper representation of the security engineering discipline during the system development effort. BridgeSec, with its Vulnerable Asset concept – which is used for binding related domain concepts into manageable entities – significantly generalises and expands the VERDICT approach. Our implementation allows security-related details to be adapted and refined as well as to accommodate information from existing body of knowledge repositories, while fully supporting ontology-based reasoning.

Navas et al. [97] attempt to address the integration of security engineering and systems engineering efforts when developing systems by using a model-based approach. However, their solution necessitates close, ongoing interaction between systems engineering and security engineering, and it does not support vulnerability management at any level. The TRADES model-based methodology partially addresses this integration of efforts, but, until now, only provided means to communicate threats and not vulnerabilities [17]. BridgeSec is unique in its ability to systematically and formally integrate the ongoing efforts of security engineering and of systems engineering in system development contexts, while allowing for loosely coupled efforts by each discipline.

CYBOK (Cybersecurity Body of Knowledge) is an algorithmic approach to vulnerability exploration [28]. It relies on specifying free-form, descriptive keywords to characterise any element of its formal model, whether a component or a semantically meaningless dependency between two components. In comparison, we explicitly codify and employ some of the domain ontology of vulnerability management using our metamodel: vulnerabilities in various levels of abstraction, supported by our "refines" relations, assigning component types to components, and associating vulnerabilities with affected component types. In CYBOK, vulnerabilities (CWEs and CVEs) are associated with the system design by a search engine that looks for a match between the descriptive keywords (of the system design) to information independently extracted from the vulnerabilities. This does not provide for a good separation of concerns between systems engineers and security engineers. We provide a more formalised structure of model elements, employing the static semantics of our metamodel to relate and reason about these elements. Furthermore, CYBOK does not address the existence of security controls within the system design or the design space, as potential mitigation. Ignoring the existence of security controls leads to analyses that ignore mitigated vulnerabilities, and, as such, can lead to many false positive alerts of vulnerabilities in the design, ultimately deeming such analyses as impractical.

The Meta Attack Language (MAL) is a domain-specific meta-language to support the further design of modelling languages, with

Table 1
Comparison of BridgeSec and related works.

Criterion	Type of methodology	Perspectives	System and security experts collaboration	Support for vulnerability management concepts	Reasoning mechanism	User-defined typing mechanism	Implementation availability
Avizieniset al. [92]	Taxonomy	System and security	Not systemised	High level definition of concepts	None	Out of scope	No
Fenz & Ekelhart [93]	Formal ontology	Security	Out of scope	Vulnerability + Component type + Control	Partial (infrastructure)	Out of scope	No
Oliveiraet al. [94]	Formal ontology	Security	Out of scope	Vulnerability + Control	None	Out of scope	No
Valenzaet al. [33]	Formal model + Attack graph	Security	Out of scope	No	Offline	Not supported	Yes
Eckhartet al. [95]	Model based + Formal ontology + Attack graph	System and security	Not systemised	Vulnerability + Component type (Limited to CPE)	None	Not supported	Yes
OnToRisk [61]	Formal ontology	Security	Out of scope	Vulnerability	Limited	Not supported	Yes
Secure tropos [96]	Informal modelling (diagrammatic)	System and security	Not systemised, tightly coupled	Control	None	Out of scope	No
VERDICT [72]	Model-based + Formal ontology + Attack defence tree	System and security	Disjoint	Vulnerability + Component type + Control	Offline, based on formal ontology queries (rigorous)	Not supported	Yes
Navaset al. [97]	Model-based	System and security	Not systemised, tightly coupled	Control	None	Out of scope	Yes
CYBOK [28]	Formal model	System and security	Not systemised	Vulnerability + Component type	Offline, based on search queries	By keywords (free-form) for: component types and vulnerabilities	Yes
MAL [34]	Formal model	Security	Out of scope	Component type + Control	Theoretical (subject to implementations)	By declarations for: component types and security controls	Partial
BridgeSec (this work)	Model-based + Formal Model	System and security	Systemised, loosely coupled	Vulnerability + Component type + Control	Online, based on formal definitions (rigorous)	By model elements (structured) for: component types, rules, security controls and vulnerabilities	Yes

the potential of automatically generating attack graphs from system architecture models (provided the concrete modelling languages correctly implement the MAL specification) [34]. Similarly to BridgeSec, MAL is formalised using set notation. It identifies the concepts of *Asset* and *Defense*, which are aligned with BridgeSec's *Component type* and *Control* concepts, respectively. However, MAL does not explicitly include the concept of *Vulnerability*, which is only implicitly addressed by the concept of *Attack step*.

Table 1 summarises key aspects of the comparative analysis between related work and BridgeSec. It shows that while some works support automated reasoning for vulnerability assessment, only a few of them allow user-specified typing of related concepts – namely vulnerabilities and the component types to which they apply. Also, our prototype implementation is the only one to provide online/on-the-fly assessment, based on elements within the model as they are created and updated. This was demonstrated throughout Section 4 when detailing the examples. Furthermore, while several other related works acknowledge the need for an integrative systems engineering and security engineering effort in designing for security, none of them explicitly systemise the interface between the disciplines and allows for manageable, loosely coupled separation of concerns. The explicit systemisation of the interface between disciplines is the unique contribution and the distinguishing aspect of BridgeSec. This systemisation underpins the potential use of BridgeSec as a framework for security by design processes, as illustrated in Fig. 3.

6. Discussion

Designing the security aspects of systems requires collaboration between systems engineering and security engineering. However, until now, the joint effort has not been well articulated into a systematic method. In this article, we propose a conceptual interface – BridgeSec – that allows for effective and systematic coordination between the two disciplines. This interface tackles significant technical management gaps related to the responsibility and coordination of the security of systems, which were independently established in a comprehensive, systematic study [36].

BridgeSec relies on our identification of an information-exchange interface that can be used to coordinate the systems engineering and security engineering efforts. The interface is purposefully bounded to two concepts – component types and controls – allowing for a divide-and-conquer strategy for the system's security design. This approach allows each of the two disciplines to work based on its own expertise and relevant information as well as supports the integration between the disciplines. Specifically, it allows security engineers to codify security-related knowledge and expectations – e.g., security policies – as well as integrate information from both internal and external sources and project it on systems designs provided – in appropriate level of details – by systems engineers.

Furthermore, BridgeSec builds on concepts pertinent to systems vulnerability management to provide a reasoning mechanism. Employing the reasoning mechanism results in a security posture assessment, which can then be communicated between systems engineering and security engineering. BridgeSec therefore facilitates informed, harmonised decision-making in support of designing systems for security.

We provided the formalisation of the vulnerability management concepts and the reasoning mechanism of BridgeSec. Moreover, we provided a prototype for BridgeSec, which is available as an open-source package. The prototype relies on a well-defined, executable

metamodel as well as model-based representations that exercise the formally defined reasoning mechanism, to provide automated inference of vulnerable components within a system's design. Our implementation extends the previous state-of-the-art TRADES methodology and tool by adding a type system for the components, additional vulnerability management-related concepts, a reasoning mechanism and a dedicated representation. All of these provide a concrete, integrated realisation of BridgeSec. The availability of a software implementation of BridgeSec – as an extension of a TRADES modelling tool – validates the soundness of its definitions and its technical validity. This does not limit the applicability of BridgeSec to TRADES models, as the conceptual information-exchange interface is agnostic to the modelling language implementation. Furthermore, model-to-model transformations can be explored to translate system designs captured in other modelling languages (e.g., SysML) into TRADES, in which our BridgeSec prototype implementation can then be exercised.

We illustrated the use of BridgeSec in two generalised, realistic scenarios. The first shows the use of BridgeSec in the early stages of systems design, while the second demonstrates the ongoing use of BridgeSec to reflect design issues and solutions even after a system has been deployed and when considering new development blocks or revisions to the system.

BridgeSec provides a vulnerability management mechanism that enables system engineers to identify vulnerable components and integrate security controls deemed appropriate as mitigation for specific vulnerabilities by security experts. Moreover, it can serve as a valuable tool for security experts to articulate and enforce comprehensive security guidelines and policies, enabling them to shape the overall security posture of the system effectively and throughout the system development life cycle.

In the first illustrative scenario, an explicit policy by a security expert is used to guide the system design and, specifically, the incorporation of security controls that need to be embedded within the design as security-oriented functionality. The policy is stated in an implementation-agnostic manner, which facilitates its use across designs and development efforts. This provides organisations – including those responsible for developing systems, for procuring systems and for regulating systems development – the ability to specify policies for different types of components (e.g., internally developed applications vs. externally developed/off-the-shelf applications).

In the second illustrative scenario, a security expert curates vulnerability information and respective security controls. This allows for informed decision-making by the development teams (e.g., a selection between applying a relevant security patch and disabling services based on system/component functionality). The decision, in turn, is communicated explicitly in the BridgeSec information model, and a reasoning mechanism supports assessing the resulting security posture of the system.

By facilitating the management of security-related information throughout the system development life cycle, BridgeSec addresses the challenges of incorporating security considerations into systems development efforts, bridging the system engineering and security engineering disciplines. This proactive, security by design approach is expected to enhance the overall security posture of systems, reducing the likelihood of security breaches and instilling greater user trust. Future empirical studies and user feedback can strengthen and expand the scope of validity. Specifically, we plan to employ BridgeSec to coordinate security in the development of diverse systems, in different domains.

We are fully aware that both security information knowledge and systems designs continue to evolve. This evolution includes an element of uncertainty as new vulnerabilities are disclosed and as new component types emerge. We also acknowledge that different domains and/or organisations may adopt and tailor diverse policies as well as focus on various types of components as the constituents of their systems designs. Our conceptual interface as well as its meta-level codification accommodate these aspects of evolution, uncertainty and adaptability and allow for great expressivity. Users can instantiate different models, including different component typing systems and rule systems, to create shared knowledge bases. For example, our metamodel (which is a codification of the BridgeSec interface) allows systems engineers to detail component types at any level of abstraction. Since a component may be associated with multiple component types, component types can even be used to specify roles of the component alongside characteristics (e.g., "Sensitive information handling application" component type alongside "Internally developed application" component type; and both can be associated with a specific software application, resulting in a need to address the union of the sets of vulnerabilities associated with each component type).

Also, as opposed to other hard-coded reasoning mechanisms (such as the one in VERDICT), our automated reasoning mechanism is designed to operate based on meta-level entities. Various organisations – specifically, security experts in these organisations – can detail different policies and/or rule systems, and system engineers can use the automated reasoning mechanism of our approach to assess design with respect to these policies/rules (specified by the security experts). The meta-level characteristic allows our conceptual interface and approach to stay up to date with evolving information security knowledge. E.g., if a future revision of NIST SP800-53 identifies new security controls, these can be instantiated and binded – by the security expert and using our user-specified rule system – with the vulnerabilities they mitigate and the component type context to which they apply. The ability to accommodate for the evolution of the vulnerability landscape – i.e., to assess systems designs against newly disclosed vulnerabilities – is already established (Section 4.2).

Further automation can streamline and enhance the efficiency of BridgeSec, making it more accessible and scalable. For example, information regarding vulnerabilities can be regularly – potentially even automatically – imported from relevant repositories (e.g., CVE records), reflecting potential vulnerabilities on system designs as soon as these vulnerabilities are disclosed, effectively transforming the BridgeSec prototype implementation into a sort of security-oriented digital twin. Future research may also attempt to automate the creation of vulnerable asset definitions. For example, one can automatically extract CVE vulnerabilities with the CWE weaknesses they refine, the CPE component types they affect and even with potential security controls, based on the information available in a CVE record; and bind these as vulnerable asset definitions. One can also extract components and component types from system models (e.g., those specified using System Modelling Language or Architecture Analysis & Design Language) or from design documents (e.g., natural language processing techniques).

Extending the reasoning mechanism to include additional aspects – such as risk management – can also contribute to the utility of BridgeSec. Specifically, the integration with OntoRisk [61], to take into consideration the risk-related concepts, can improve the decision support capabilities of BridgeSec. Also, for practical applications, the reasoning mechanism should be expanded to support hierarchical considerations of the component types and vulnerabilities. For example, if a Windows Server component includes – by design – a control that mitigates a CWE, the reasoning mechanism can infer that the component – regardless of the specific Windows Server version implemented – is not vulnerable to CVEs that refine the specific CWE and are associated with such servers.

In conclusion, BridgeSec represents a significant step towards integrating security considerations into system engineering processes. By providing a disciplined and practical formal approach to security information management, it enables concise communication of codified knowledge and structured information between relevant stakeholders, and, consequently, effective vulnerability management and security by design. These provide a strong basis for rigorous, collaborative reasoning about the security posture of systems designs.

Further research, evaluation, and refinement of the methodology will seek to enhance its capabilities and broaden its applicability in diverse system development contexts. Also, we are working to elaborate the reasoning mechanism, taking into account additional aspects such as the hierarchical placement of system components and security controls, refinement relations of the type system and the refinement relations between conceptual and concrete vulnerabilities.

CRediT authorship contribution statement

Avi Shaked: Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Nan Messe:** Writing – original draft, Visualization, Resources, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors wish to thank Professor Tom Melham for his valuable advice. This work is supported by the ICO, Cybersecurity Institute of Occitanie, funded by the Occitanie Region, France, and by Innovate UK (grant #75243).

Data availability

Research data and code is shared publicly in an online repository, under a permissive open source license.

References

[1] Thames L, Schaefer D. Cybersecurity for industry 4.0: Analysis for design and manufacturing. 1st ed. Springer series in advanced manufacturing, Springer; 2017.
[2] Parnell GS, Driscoll PJ, Henderson DL. Decision making in systems engineering and management. Wiley series in systems engineering and management, Wiley; 2008.
[3] Bu F, Wang N, Jiang B, Liang H. "Privacy by Design" implementation: Information system engineers' perspective. Int J Inf Manage 2020;53:102124.
[4] Beznosov K, Chess B. Security for the rest of us: An industry perspective on the secure-software challenge. IEEE Softw 2008;25(1):10–2. http://dx.doi.org/10.1109/MS.2008.18.
[5] Ross R, McEvilley M, Oren J. NIST SP 800-160 volume 1 rev 1: Engineering trustworthy secure systems. National Institute of Standards and Technology; 2022, http://dx.doi.org/10.6028/NIST.SP.800-160v1r1.
[6] Lowry PB, Dinev T, Willison R. Why security and privacy research lies at the centre of the information systems (IS) artefact: Proposing a bold research agenda. Eur J Inf Syst 2017;26(6):546–63.
[7] Ross R, Pillitteri V, Graubart R, Bodeau D, McQuaid R. NIST SP 800-160 volume 2: Developing cyber-resilient systems: A systems security engineering approach. National Institute of Standards and Technology; 2021.
[8] Mailloux LO, McEvilley M, Khou S, Pecarina JM. Putting the "systems" in security engineering: An examination of NIST special publication 800-160. IEEE Secur Priv 2016;14(4):76–80. http://dx.doi.org/10.1109/MSP.2016.77.
[9] Ray LL. Security considerations for the spiral development model. Int J Inf Manage 2013;33(4):684–6.
[10] Steward Jr C, Wahsheh LA, Ahmad A, Graham JM, Hinds CV, Williams AT, et al. Software security: The dangerous afterthought. In: 2012 ninth international conference on information technology - new generations. 2012, p. 815–8. http://dx.doi.org/10.1109/ITNG.2012.60.
[11] Olivero MA, Bertolino A, Dominguez-Mayo FJ, Escalona MJ, Matteucci I. Security assessment of systems of systems. In: 2019 IEEE/ACM 7th international workshop on software engineering for systems-of-systems and 13th workshop on distributed software development, software ecosystems and systems-of-systems. WDES, 2019, p. 62–5. http://dx.doi.org/10.1109/SESoS/WDES.2019.00017.
[12] Kreitz M. Security by design in software engineering. SIGSOFT Softw Eng Notes 2020;44(3):23. http://dx.doi.org/10.1145/3356773.3356798.
[13] CISA, et al. Shifting the balance of cybersecurity risk: Principles and approaches for security-by-design and -default. Tech. rep., Cybersecurity and Infrastructure Security Agency; 2023.
[14] Nguyen PH, Ali S, Yue T. Model-based security engineering for cyber-physical systems: A systematic mapping study. Inf Softw Technol 2016;83:116–35.
[15] Guggenmos F, Häckel B, Ollig P, Stahl B. Security first, security by design, or security pragmatism — strategic roles of IT security in digitalization projects. Comput Secur 2022;118(C). http://dx.doi.org/10.1016/j.cose.2022.102747.
[16] Porter SJ. Cyber security by design vs. Post deployment hardening. Tech. rep., Livermore, CA (United States): Lawrence Livermore National Lab. (LLNL); 2017.
[17] Shaked A. A model-based methodology to support systems security design and assessment. J Ind Inf Integr 2023;33:100465. http://dx.doi.org/10.1016/j.jii.2023.100465.
[18] Spiekermann S, Korunovska J, Langheinrich M. Inside the organization: Why privacy and security engineering is a challenge for engineers. Proc IEEE 2019;107(3):600–15. http://dx.doi.org/10.1109/JPROC.2018.2866769.
[19] Messe N, Chiprianov V, Belloir N, El-Hachem J, Fleurquin R, Sadou S. Asset-oriented threat modeling. In: 2020 IEEE 19th international conference on trust, security and privacy in computing and communications. IEEE; 2020, p. 491–501.
[20] Bakirtzis G, Ward G, Deloglos C, Elks C, Horowitz B, Fleming C. Fundamental challenges of cyber-physical systems security modeling. In: 2020 50th annual IEEE-iFIP international conference on dependable systems and networks-supplemental volume. DSN-s, IEEE; 2020, p. 33–6.
[21] Haney JM, Lutters WG. "It's Scary...It's Confusing...It's Dull": How Cybersecurity Advocates Overcome Negative Perceptions of Security. In: Proceedings of the fourteenth USeNIX conference on usable privacy and security. SOUPS '18, USA: USENIX Association; 2018, p. 411–25.
[22] de Vries S. Embedding security by design: A shared responsibility. 2023, https://www.darkreading.com/application-security/embedding-security-by-design-a-shared-responsibility-. [Accessed 14 June 2023].
[23] Azzazi A, Shkoukani M. A knowledge-based expert system for supporting security in software engineering projects. Int J Adv Comput Sci Appl 2022;13(1):18.
[24] Hilbrich M, Frank M. Enforcing security and privacy via a cooperation of security experts and software engineers: A model-based vision. In: 2017 IEEE 7th international symposium on cloud and service computing. IEEE; 2017, p. 237–40.
[25] Haberfellner R, de Weck O, Fricke E, Vössner S. Systems engineering: Fundamentals and applications. Springer International Publishing; 2019.
[26] Walden DD, Roedler GJ, Forsberg K. INCOSE systems engineering handbook version 4: Updating the reference for practitioners. In: INCOSE international symposium, vol. 25. (1):Wiley Online Library; 2015, p. 678–86.
[27] Friedenthal S, Moore A, Steiner R. A practical guide to SysML, third edition: The systems modeling language. 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2014.
[28] Bakirtzis G, Simon BJ, Collins AG, Fleming CH, Elks CR. Data-driven vulnerability exploration for design phase system analysis. IEEE Syst J 2020;14(4):4864–73.
[29] Anderson RJ. Security engineering: A guide to building dependable distributed systems. 2nd ed. Wiley Publishing; 2008.
[30] Matulevičius R. Fundamentals of secure system modelling. Springer International Publishing; 2017.
[31] Schneier B. Secrets and Lies, Chapter 21: Attack Trees. John Wiley & Sons, Ltd; 2015, p. 318–33. http://dx.doi.org/10.1002/9781119183631.ch21.
[32] Xiong W, Lagerström R. Threat modeling — A systematic literature review. Comput Secur 2019;84(C):53–69. http://dx.doi.org/10.1016/j.cose.2019.03.010.
[33] Valenza F, Karafili E, Steiner RV, Lupu EC. A hybrid threat model for smart systems. IEEE Trans Dependable Secure Comput 2022;1–14. http://dx.doi.org/10.1109/TDSC.2022.3213577.
[34] Hacks WWS, Ekstedt M, Johnson P, Lagerström R. The meta attack language — A formal description. Comput Secur 2023;130:103284. http://dx.doi.org/10.1016/j.cose.2023.103284.
[35] Donalds C, Osei-Bryson K-M. Cybersecurity compliance behavior: Exploring the influences of individual decision style and other antecedents. Int J Inf Manage 2020;51:102056.
[36] Olivero MA, Bertolino A, Dominguez-Mayo FJ, Escalona MJ, Matteucci I. A systematic mapping study on security for systems of systems. Int J Inf Secur 2024;23(2):787–817.
[37] Yskout K, Heyman T, Van Landuyt D, Sion L, Wuyts K, Joosen W. Threat modeling: From infancy to maturity. In: Proceedings of the ACM/IEEE 42nd international conference on software engineering: New ideas and emerging results. 2020, p. 9–12.
[38] Flechais I, Sasse A. Stakeholder involvement, motivation, responsibility, communication: How to design usable security in e-science. Int J Hum-Comput Stud 2009;67:281–96. http://dx.doi.org/10.1016/j.ijhcs.2007.10.002.
[39] De Vasconcelos JB, Kimble C, Carreteiro P, Rocha Á. The application of knowledge management to software evolution. Int J Inf Manage 2017;37(1):1499–506.
[40] Akgün AE. Team wisdom in software development projects and its impact on project performance. Int J Inf Manage 2020;50:228–43.
[41] Durugbo C, Tiwari A, Alcock JR. Modelling information flow for organisations: A review of approaches and future challenges. Int J Inf Manage 2013;33(3):597–610.

[42] Samonas S, Dhillon G, Almusharraf A. Stakeholder perceptions of information security policy: Analyzing personal constructs. Int J Inf Manage 2020;50:144–54.
[43] Messe N, Belloir N, Chiprianov V, El-Hachem J, Fleurquin R, Sadou S. An asset-based assistance for secure by design. In: 2020 27th Asia-Pacific software engineering conference. APSEC, IEEE; 2020, p. 178–87.
[44] TRADES Tool. online repository, https://github.com/IAI-Cyber/TRADES/. [Accessed 15 June 2023].
[45] MITRE. Common weakness enumeration, https://cwe.mitre.org/. [Accessed 21 June 2023].
[46] MITRE. Common vulnerabilities and exposures, https://www.cve.org/. [Accessed 21 June 2023].
[47] MITRE. Common platform enumeration, https://cpe.mitre.org/. [Accessed 21 June 2023].
[48] Security and privacy controls for information systems and organizations. 2020, Special Publication (NIST SP), National Institute of Standards and Technology NIST-SP 800-53.
[49] MITRE. Common attack pattern enumeration and classification, https://capec.mitre.org/. [Accessed 21 October 2024].
[50] MITRE.
[51] Howard M, Lipner S. The security development lifecycle, vol. 8. Microsoft Press Redmond; 2006.
[52] Messe NZ. security by design: an asset-based approach to bridge the gap between architects and security experts (Ph.D. thesis), Université de Bretagne Sud; 2021.
[53] Dissanayake N, Jayatilaka A, Zahedi M, Babar MA. Software security patch management — A systematic literature review of challenges, approaches, tools and practices. Inf Softw Technol 2022;144:106771. http://dx.doi.org/10.1016/j.infsof.2021.106771.
[54] Tavčar J, Horváth I. A review of the principles of designing smart cyber-physical systems for run-time adaptation: Learned lessons and open issues. IEEE Trans Syst Man Cybern A 2019;49(1):145–58. http://dx.doi.org/10.1109/TSMC.2018.2814539.
[55] Van Den Berghe A, Yskout K, Scandariato R, Joosen W. A lingua franca for security by design. In: 3rd annual IEEE cybersecurity development conference. IEEE COMPUTER SOC; 2018, p. 69–76. http://dx.doi.org/10.1109/SecDev.2018.00017.
[56] Cavoukian A, Dixon M. Privacy and security by design: An enterprise architecture approach. Canada: Information and Privacy Commissioner of Ontario; 2013.
[57] Shukla A, Katt B, Nweke LO, Yeng PK, Weldehawaryat GK. System security assurance: A systematic literature review. Comp Sci Rev 2022;45:100496. http://dx.doi.org/10.1016/j.cosrev.2022.100496.
[58] Casola V, De Benedictis A, Rak M, Villano U. A novel security-by-design methodology: Modeling and assessing security with a quantitative approach. J Syst Softw 2020;163:110537. http://dx.doi.org/10.1016/j.jss.2020.110537.
[59] Mellado D, Blanco C, Sánchez LE, Fernández-Medina E. A systematic review of security requirements engineering. Comput Stand Interfaces 2010;32(4):153–65. http://dx.doi.org/10.1016/j.csi.2010.01.006.
[60] Houmb SH, Franqueira VNL, Engum EA. Quantifying security risk level from CVSS estimates of frequency and impact. J Syst Softw 2010;83(9):1622–34. http://dx.doi.org/10.1016/j.jss.2009.08.023, Software Dependability.
[61] Shaked A, Margalit O. Sustainable risk identification using formal ontologies. Algorithms 2022;15(9):316.
[62] Mead NR, Stehney T. Security quality requirements engineering (SQUARE) methodology. SIGSOFT Softw Eng Notes 2005;30(4):1–7. http://dx.doi.org/10.1145/1082983.1083214.
[63] Fabian B, Gurses S, Heisel M, Santen T, Schmidt H. A comparison of security requirements engineering methods. Requir Eng 2010;15:7–40. http://dx.doi.org/10.1007/s00766-009-0092-x.
[64] Wach P, Salado A. Model-based security requirements for cyber-physical systems in SysML. In: 2020 IEEE systems security symposium. SSS, 2020, p. 1–7. http://dx.doi.org/10.1109/SSS47320.2020.9174222.
[65] Saini V, Duan Q, Paruchuri V. Threat modeling using attack trees. J Comput Sci Coll 2008;23(4):124–31.
[66] Byres EJ, Franz M, Miller D. The use of attack trees in assessing vulnerabilities in SCADA systems. In: IEEE conf. international infrastructure survivability workshop. institute for electrical and electronics engineers. 2004.
[67] Mauw S, Oostdijk M. Foundations of attack trees. In: Won DH, Kim S, editors. Information security and cryptology - ICISC 2005. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006, p. 186–98.
[68] Pinchinat S, Acher M, Vojtisek D. ATSyRa: An integrated environment for synthesizing attack trees. In: Mauw S, Kordy B, Jajodia S, editors. Graphical models for security. Cham: Springer International Publishing; 2016, p. 97–101.
[69] Kordy B, Mauw S, Radomirovic S, Schweitzer P. Attack-defense trees. J Logic Comput 2014;24(1):55–87. http://dx.doi.org/10.1093/logcom/exs029.
[70] Donner M. Toward a security ontology. IEEE Secur Priv 2003;1(3):6–7.
[71] Oltramari A, Cranor LF, Walls RJ, Mcdaniel P. Building an ontology of cyber security. In: Semantic technologies for intelligence, defense, and security. 2014.
[72] Meng B, Larraz D, Siu K, Moitra A, Interrante J, Smith W, et al. Verdict: A language and framework for engineering cyber resilient and safe system. Systems 2021;9(1):18.
[73] Selic B. The pragmatics of model-driven development. IEEE Softw 2003;20(5):19–25. http://dx.doi.org/10.1109/MS.2003.1231146.
[74] Czarnecki K, Helsen S. Classification of model transformation approaches. In: Proceedings of the 2nd OOPSLA workshop on generative techniques in the context of the model driven architecture, vol. 45. (3):USA; 2003, p. 1–17.
[75] Nguyen P, Kramer M, Klein J, Le Traon Y. An extensive systematic review on the model-driven development of secure systems. Inf Softw Technol 2015;68:62–81. http://dx.doi.org/10.1016/j.infsof.2015.08.006.
[76] Van Den Berghe A, Scandariato R, Yskout K, Joosen W. Design notations for secure software: A systematic literature review. Softw Syst Model 2017;16(3):809–31. http://dx.doi.org/10.1007/s10270-015-0486-9.
[77] Shaked A. Modeling for rapid systems prototyping: Hospital situational awareness system design. Systems 2021;9(1):12. http://dx.doi.org/10.3390/systems9010012.
[78] Shaked A. PROVE tool: A tool for designing and analyzing process descriptions. Softw Impacts 2022;12:100234. http://dx.doi.org/10.1016/j.simpa.2022.100234.
[79] Matulevicius R, Dumas M. Towards model transformation between SecureUML and UMLsec for role-based access control. In: Barzdins J, Kirikova M, editors. Databases and information systems VI - selected papers from the ninth international baltic conference, dB&iS 2010, July 5-7, 2010, riga, latvia. Frontiers in artificial intelligence and applications, vol. 224, IOS Press; 2010, p. 339–52. http://dx.doi.org/10.3233/978-1-60750-688-1-339.
[80] Roudier Y, Apvrille L. SysML-Sec: A model driven approach for designing safe and secure systems. In: 2015 3rd international conference on model-driven engineering and software development. MODELSWARD, 2015, p. 655–64.
[81] Mili S, Nguyen N, Chelouah R. Model-driven architecture based security analysis. Syst Eng 2021;24(5):307–21. http://dx.doi.org/10.1002/sys.21581.
[82] Hachem JE, Khalil TA, Chiprianov V, Babar A, Aniorte P. A model driven method to design and analyze secure architectures of systems-of-systems. In: 2017 22nd international conference on engineering of complex computer systems. ICECCS, 2017, p. 166–9. http://dx.doi.org/10.1109/ICECCS.2017.31.
[83] Rouland Q, Hamid B, Jaskolka J. A model-driven formal methods approach to software architectural security vulnerabilities specification and verification. J Syst Softw 2025;219:112219.
[84] Shaked A. Digital modeling of a domain ontology for hospital information systems. In: Fred A, Aveiro D, Dietz J, Salgado A, Bernardino J, Filipe J, editors. Knowledge discovery, knowledge engineering and knowledge management. Cham: Springer International Publishing; 2022, p. 157–66.
[85] Shaked A. Facilitating the integrative use of security knowledge bases within a modelling environment. J Cybersecur Priv 2024;4(2):264–77.
[86] Kotonya G, Sommerville I. Requirements engineering — Processes and techniques. John Wiley & Sons; 1998.
[87] CVE-2019-1181 vendor advisory, https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2019-1181. [Accessed 1 June 2023].
[88] CVE-2019-1182 NIST national vulnerability database, https://nvd.nist.gov/vuln/detail/cve-2019-1182. [Accessed 1 June 2023].
[89] CVE-2019-1222 NIST national vulnerability database, https://nvd.nist.gov/vuln/detail/cve-2019-1222. [Accessed 1 June 2023].
[90] CVE-2019-1226 NIST national vulnerability database, https://nvd.nist.gov/vuln/detail/cve-2019-1226. [Accessed 1 June 2023].
[91] CVE-2020-0655 vendor advisory, https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2020-0655. [Accessed 1 June 2023].
[92] Avizienis A, Laprie J-C, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secure Comput 2004;1(1):11–33. http://dx.doi.org/10.1109/TDSC.2004.2.
[93] Fenz S, Ekelhart A. Formalizing information security knowledge. In: Proceedings of the 4th international symposium on information, computer, and communications security. ASIACCS '09, New York, NY, USA: Association for Computing Machinery; 2009, p. 183–94. http://dx.doi.org/10.1145/1533057.1533084.
[94] Oliveira Í, Sales TP, Baratella R, Fumagalli M, Guizzardi G. An ontology of security from a risk treatment perspective. In: International conference on conceptual modeling. Springer; 2022, p. 365–79.
[95] Eckhart M, Ekelhart A, Weippl E. Automated security risk identification using automationml-based engineering data. IEEE Trans Dependable Secure Comput 2022;19(03):1655–72. http://dx.doi.org/10.1109/TDSC.2020.3033150.
[96] Mouratidis H, Giorgini P, Manson G. Integrating security and systems engineering: Towards the modelling of secure information systems. In: Eder J, Missikoff M, editors. Advanced information systems engineering. Berlin, Heidelberg: Springer Berlin Heidelberg; 2003, p. 63–78.
[97] Navas J, Voirin J-L, Paul S, Bonnet S. Towards a model-based approach to systems and cyber security co-engineering. INCOSE Int Symp 2019;29(1):850–65. http://dx.doi.org/10.1002/j.2334-5837.2019.00639.x.